Error-correcting Codes for Data Storage in DNA

Yuanyuan Tang

Department of Electrical and Computer Engineering
University of Virginia, Charlottesville, USA
yt5tz@virginia.edu

A dissertation presented to
the Faculty of the School of Engineering and Applied Sciences
at the
University of Virginia

In Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy
in
Electrical Engineering

Abstract

Recent advances in DNA sequencing, synthesis, and editing technologies have made DNA a promising alternative to conventional storage media. Compared to traditional media, DNA has the advantages of high data density, longevity, and ease of copying information. However, a diverse set of errors, including deletions, insertions, substitutions, and duplications may arise at different stages of the data storage and retrieval process. The dissertation focuses on constructing error-correcting codes to correct multiple sources of error in DNA data storage.

First, duplications and various types of edit errors (including insertions, deletions, and substitutions) may affect the integrity of data stored in DNA. While recent research has addressed correcting each of these error types in isolation, there is a notable gap in correcting them simultaneously, to the best of our knowledge. This is problematic as the presence of one type of error does not preclude another, especially in long sequences. In this dissertation, we present families of codes that simultaneously correct any number of duplications and a bounded number of edit errors, with small impact on code rate compared to the state-of-the-art codes for duplications only.

Second, to correct various types of localized errors, we construct error-correcting codes for substring edit errors. A localized error occurs when several errors occur in a bounded window of the input data. In the literature, localized deletions, insertions, and substitutions have been studied. A substring edit, defined as replacing a substring with another string, both with bounded length, encompasses all aforementioned localized errors. In this work, we first show using statistical analysis of errors that substring edits are common and viewing errors as substring edits in principle will require less redundancy. Then error-correcting codes are constructed to correct a substring edit with low redundancy, providing a universal solution to correct a diverse set of localized errors.

Third, we develop codes for improving the reliability of an emerging DNA synthesis method that due its cost-effectiveness can alleviate one of the main obstacles for wide spread use of data storage in DNA, namely *terminator-free template-independent* enzymatic DNA synthesis (simplified as enzymatic synthesis). While more economical, enzymatic synthesis suffers from a higher error rate. Specifically, the number of times each base is synthesized cannot be controlled precisely and also deletion errors are likely. Existing encoding methods have either a *writing rate* upper bounded by $\log_2 3$ bits per unit time or cannot combat against deletions. In this dissertation, we present an error-correcting code and a decoding algorithm to combat deletions and achieve a writing rate higher than the state of the art. The error probability of the proposed method is analyzed and the strategies for tuning the parameters to achieve desirable tradeoffs between different requirements are presented.

Together, all the error correction algorithms, tools, and techniques for duplications and edits, substring edits, and the errors from the enzymatic synthesis presented in this work have the potential to contribute to the development of DNA data storage systems with increased capacity, higher reliability, and lower cost.

Acknowledgement

First, I would like to thank my advisor Prof. Farzad Farnoud. During my journey of pursuing my Ph.D. degree, I got a lot of training in doing research, writing academic papers, and presenting works. His rigorous attitude towards research and academic works will benefit me in my long-term career. I was also provided with many opportunities to cooperate with other researchers, which extended my skills and connections. Without his support, it would be impossible for me to have a chance to defend my dissertation. I appreciate the help and insightful suggestions from Prof. Nikolaos Sidiropoulos, Prof. Stephen G. Wilson, Prof. Cong Shen, and Prof. Tom Fletcher in my dissertation committee about my dissertation and presentation. I also appreciate the help I have gotten from all the professors and teachers during and before my Ph.D. program. In addition, I would like to express my gratitude to all my coauthors, including Dr. Ryan Gabrys, Dr. Yonatan Yehezkeally, Dr. Moshe Schwartz, Dr. Hao Lou, Mr. Shuche Wang, and Mr. Yuting Li.

I would also like to thank all my labmates including Hao Lou, Tao Jin, Kallie Whritenour, Sarvin Motamen, and Yuting Li. You have supported me and helped me at various stages of my Ph.D. studies, including the qualifying exam and the proposal. Moreover, the group brainstorming sessions have significantly broadened my understanding and offered invaluable insights towards resolving my research challenges.

I appreciate the support of all my friends during my study at UVa, including Zhelong He, Shuo Li, Jian Wang, Chuanhao Li, Jiarui Xin, Shili Sheng, Yinzhu jin, and many others. Without them, I would not have had a wonderful journey at UVa and gone through all the tough time during my study. Even though most of them have moved to other places, I will always treasure the friendships with them.

Finally, I want to express my deep gratitude to my family members, especially my parents. I was born in a small village, where almost all people have been farmers for generations. As the only child and the first one to attend university, my parents did not ask me to help financially support the family as long as I completed my Master's program. They also supported me to pursue my academic dreams in the USA. Furthermore, their optimistic attitudes toward challenges in life has been a constant source of inspiration. Thanks to their unwavering encouragement and support, I have successfully overcome numerous challenges and am now poised to defend my dissertation.

Yuanyuan Tang, November, 2023

Publications

Journals

- S. Wang, Y. Tang, J. Sima, *et al.*, "Non-binary codes for correcting a burst of at most t deletions", *IEEE Transactions on Information Theory*, 2023 (accepted)

  Y. Tang, S. Wang, H. Lou, *et al.*, "Low-redundancy codes for correcting multiple short-duplication and edit errors", *IEEE Transactions on Information Theory*, vol. 69, no. 5, pp. 2940–2954, 2023

- Y. Tang and F. Farnoud, "Error-correcting codes for short tandem duplication and edit errors", *IEEE Transactions on Information Theory*, vol. 68, no. 2, pp. 871–880, 2022

- Y. Tang and F. Farnoud, "Error-correcting codes for noisy duplication channels", *IEEE Transactions on Information Theory*, vol. 67, no. 6, pp. 3452–3463, 2021

- Y. Tang, Y. Yehezkeally, M. Schwartz, *et al.*, "Single-error detection and correction for duplication and substitution channels", *IEEE Transactions on Information Theory*, vol. 66, no. 11, pp. 6908–6919, 2020

Conferences

- Y. Tang, S. Motamen, H. Lou, *et al.*, "Correcting a substring edit error of bounded length", in *2023 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2023, pp. 2720–2725

- Y. Tang, S. Wang, R. Gabrys, *et al.*, "Correcting multiple short-duplication and substitution errors", in *2022 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2022, pp. 1–6

- Y. Tang and F. Farnoud, "Correcting deletion errors in DNA data storage with enzymatic synthesis", in *2021 IEEE Information Theory Workshop (ITW)*, 2021, pp. 1–6

- Y. Tang, H. Lou, and F. Farnoud, "Error-correcting codes for short tandem duplications and at most p substitutions", in *2021 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2021, pp. 1835–1840

- Y. Tang and F. Farnoud, "Error-correcting codes for short tandem duplication and substitution errors", in *IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2020, pp. 734–739

- Y. Tang, Y. Yehezkeally, M. Schwartz, *et al.*, "Single-error detection and correction for duplication and substitution channels", in *2019 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2019, pp. 300–304

- Y. Tang and F. Farnoud, "Error-correcting codes for noisy duplication channels", in *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, IEEE, 2019, pp. 140–146
- S. Wang, Y. Tang, R. Gabrys, *et al.*, "Permutation codes for correcting a burst of at most t deletions", in *2022 58th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, IEEE, 2022, pp. 1–6

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Motivation and overview

In the past decade, the amount of data created and consumed worldwide has increased exponentially. For example, it is estimated that the amount of data will reach 64.2 Zettabytes (1 zettabyte = 10^{21} Bytes) in 2020, posing great challenges for conventional data storage media [27], [71]. Recent advances in deoxyribonucleic acid (DNA) sequencing, synthesis, and editing technologies [53], [90], [98] have made DNA a promising alternative to conventional storage media. Compared to traditional media, DNA has several advantages, including high data density, longevity, and ease of generating copies. For example, the DNA sequences of species extinct for $10,000$ years have been successfully recovered [98], and a single human cell contains an amount of DNA that can ideally hold 6.4 Gb of information.

Recent works [13], [22], [23], [38], [52], [59], [98], [99] have demonstrated the feasibility of DNA data storage and have led to significant advances, such as the ability to provide random-access to the data [99], a DNA data storage system with portable size [97], and a combination of an inexpensive enzymatic synthesis method with error-correcting codes to achieve low cost [13], [38].

In general, a DNA data storage pipeline consists of encoding, synthesis, storage, sequencing, and decoding, as shown in Figure 1.1. Advances in these processes have brought DNA data storage closer to practical applications.. For example, data can be stored in DNA *in vitro* or *in vivo*, where data storage *in vivo* can provide a more cost-effective replication method as well as a protective shell compared to data storage in vitro [69], [70], [102]. By using the *CRISPR/Cas system* and *Illumina sequencing*, an image and a short video has been stored in bacterial cells and recovered successfully [69]. A new sequencing technique called *Nanopore sequencing* is also promising [25] as it allows for longer reads and real-time sequencing at the cost of accuracy [25]. Because of its promising properties, DNA data storage has been proposed for various applications, including long-term data storage [30], [31], watermarking genetically-modified organisms (GMOs), and labeling organisms in biological studies.

Despite these advances, there are still significant challenges to be overcome. One obvious challenge is that a diverse set of errors may occur at different stages of the data synthesis, storing, and retrieval process, such as duplications, deletions, insertions, and substitutions. Many recent

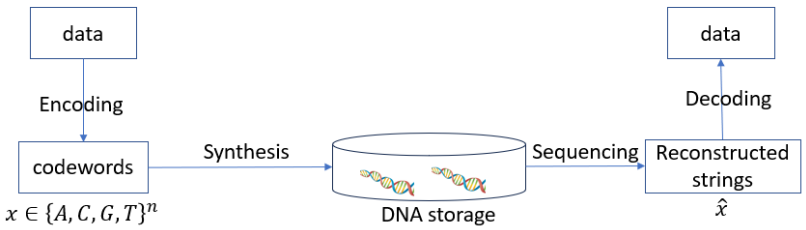


Figure 1.1: The general DNA storage model with processes of synthesis, storage, and sequencing.

works, such as [6], [8], [14], [18], [28], [30], [34], [36], [38]–[40], [56], [57], [61], [69], [76], [79], [80], [100], [101], have been devoted to fighting against these errors. For example, the authors in [56], [61] constructed error-correcting codes to correct substitution errors and meet specific GC-content balance and run-length constraints. The authors in [8], [30], [36], [43], [100] focused on constructing error-correcting codes to fight against duplication errors, and the work [96] explored the capacity of the DNA storage channels with insertions, deletions, and substitutions. Furthermore, the works in [2], [24], [42], [93] designed codes to correct a burst of insertions/deletions or localized deletions/insertions. For the new inexpensive enzymatic synthesis, the authors in [28], [38], [68] proposed algorithms or error-correcting codes to achieve low-cost data storage.

In order to provide reliable DNA data storage, by extending the previous works, this dissertation focuses on constructing error-correcting codes to fight against multiple sources of errors in DNA data storage, including i) duplications and edits, ii) localized errors, and iii) errors introduced by enzymatic synthesis.

1.2 Background and related work

Correcting errors is crucial for ensuring reliability in data storage and communication systems. One of the most common approaches involves to steps: i) modeling the input-output relationship as a communication channel with specific types of errors and ii) constructing error-correcting codes. An error-correcting code works by limiting the set of the inputs to a specific set of sequences, each called a codeword. No two codewords are likely to produce the same output as long as the errors satisfy certain requirements. Hence, from the output of the channel, we will be able to identify the correct input. In general, the error-correcting codes fight against errors by introducing extra redundancy to make the codewords distinguishable even after changes due to errors. We start by providing an example of error-correcting codes.

Example 1. *Suppose a channel model may flip one symbol of the input sequence. A simple error-correcting code is to choose strings that repeat each symbol* 3 *times, i.e.* $\mathcal{C}_3 = \{000, 111\}$ *for the input* $0, 1$*. If the codeword* 000 *is sent in the channel, the output may appear as one sequence in the set* $\{000, 100, 010, 001\}$*. Then we can derive the codeword* 000 *and the input* 0*. Similarly, the codeword* 111 *can be recovered if one of* $\{111, 011, 101, 110\}$ *appears as the channel output.*

Based on Example 1, the design of error-correcting codes and decoding algorithms can pro-

vide reliable communication. We define the following metrics to evaluate the performance of an error-correcting code. Suppose $\mathcal{C}_n$ is an error-correcting code consisting of a subset of codewords (sequences) of length n. We define the *code rate* [63] as

$$R(\mathcal{C}_n) = \frac{1}{n} \log_m \|\mathcal{C}_n\|,$$

where q is the size of alphabet size, $\|\mathcal{C}_n\| \leqslant q^n$ denotes the size of the error-correcting code, and $R(\mathcal{C}_n)$ is bounded by $\log_m q$ for $m \in \{2, q\}$ with the unit as bits/symbol and symbol/symbol respectively. We also define the *redundancy* of the code as

$$r(\mathcal{C}_n) = n - \log_q \|\mathcal{C}_n\|,$$

Then the code $\mathcal{C}_3$ in Example 1 has rate 1/3 bits/symbol and the redundancy 2 symbols. Compared to the maximum rate 1 bit/symbol, the error-correcting code may be inefficient since another two extra bits are sent in order to transmit a single bit 0 or 1.

Therefore, given a channel with errors, the core problem is to construct error-correcting codes that can correct errors with high code rate or with as little redundancy as possible. Since a diverse set of errors may occur in different stages of data storage and retrieval, this dissertation starts by reviewing the existing error-correcting codes to fight against various sources of errors in DNA data storage.

1.2.1 Duplications and edit errors

A duplication in a DNA sequence generates a copy of a substring and then inserts it directly following the original substring [30], resulting in a *tandem repeat*, where the duplication length is the length of the copy. For example, given ACTG, a tandem duplication may generate ACT$\underline{\text{CT}}$G, where the inserted copy is marked by the underline, and CT$\underline{\text{CT}}$ is a *tandem repeat*. Evidence of this process is found in genomes, such as tandem repeats generated by the *slipped-strand mispairings* [15], [37]. Correcting both fixed-length duplications with the same (duplication) length [30], [36], [79], [81], [101] and bounded-length duplications [7], [8], [29], [30], [35], [84] have been studied recently. Bounded-length duplications are those whose length is at most a given constant. In particular, duplications of length at most 3 are referred to as *short duplications*. For fixed-length duplication errors, the authors in [30] construct error-correcting codes to correct an arbitrary number of fixed-length duplications with the asymptotic optimal code rate $\log_2 q - \frac{(q-1)\log_2 e}{q^{k+1}}(1 + o(1))$ [30]. For short duplication errors, the code in [30] can correct an arbitrary number of short duplications with the highest known asymptotic rate [8], [83] by choosing the set of $\leqslant$3-*irreducible strings* with length at most n, where each $\leqslant$3-irreducible string is a string that contains no (tandem) repeats with length at most 6. The rate of the code is $\log_2 2.6590$ bits/symbol when the alphabet size q is 4 [83] and approximately $\log_2(q-1)$ as q increases [8].

We note that for $q = 4$, which is of interest in DNA storage, and for $k = 2, 3, 4$, the asymptotic rates of optimal codes correcting any number of exact fixed-length duplications can be shown to equal $1.9226, 1.9827, 1.9958$ bits/symbol with the logarithms in base 2, respectively [30]. The fact

that these values are close to 2 bits/symbol indicates that the rate penalty for correcting an infinite number of exact k-duplication errors compared to only correcting a finite number is not significant and diminishes as k grows. Furthermore, DNA storage is considered a promising long-term data storage solution that may suffer many errors. Hence, we focus on correcting many exact fixed-length duplications and bounded-length duplications rather than a finite number.

Apart from duplications, edit errors such as insertions, deletions, and substitutions are also frequently occurring in DNA storage [4], [38], [62], [78], [96]. For example, point mutations such as substitutions are observed in tandem repeat regions of the genomes [62]. Due to an arbitrary number of short duplications in the long-term storage [30], a single edit error may affect an unbounded segment of the output. In order to correct both duplications and edit errors, one simple idea is to construct error-correcting codes as the intersection of duplication-correcting codes and indel/edit-correcting codes. However, Example 32 shows the failure of this idea even for any fixed-length duplications and an extra substitution. Another idea is to consider tandem duplications as edits and then apply indel/edit-error-correcting codes to correct them. However, an arbitrary number of duplications will lead to a high proportion of edits and a low rate [4], [72]. Therefore, it motivates the design of error-correcting codes with low redundancy or high rate to simultaneously combat against many duplications and edit errors.

1.2.2 Localized errors

Localized errors are errors that cluster in windows with lengths much shorter than the whole sequence and are observed in various applications such as wireless communication, disk data storage, DNA storage, and document synchronization. The problem of burst or localized deletions has been studied by several works [2], [9], [17], [24], [42], [65], [66], [91], [93], [104]. More specifically, codes for correcting a burst of at most k substitutions were proposed in [17], [104]. Codes capable of correcting a burst of exactly k deletions were studied by [9], [65], [66], including [9], [66] over binary sequences and [65] over q-ary sequences. Furthermore, [42], [66], [93], [94] focused on correcting a burst of at most k deletions (or k insertions) while the works in [2], [24] studied localized deletions occurring in a window with bounded length k. In particular, the authors in [2], [42] construct optimal codes with redundancy roughly $\log n$ bits. Based on the *guess&Check* idea, the authors in [24] construct error-correcting codes to correct localized deletions in window size k with non-zero error probability, where the codes have redundancy of roughly $5\log n + 1$ bits when $k = o(\log n)$ and roughly $5k + 1$ bits when $k = \Omega(\log n)$.

A *substring edit error*, which replaces one substring with another string both with bounded lengths, is also considered a localized error. For example, given $\boldsymbol{x} = 0132132$, a substring edit of length 3 may generate $\boldsymbol{y} = 013\underline{12}2$ by replacing $\boldsymbol{x}_{[4:6]} = 213$ with $\boldsymbol{y}_{[4:5]} = 12$. In particular, burst deletions/insertions/substitutions and deletions/insertions occurring in a bounded window can all be considered as special cases of substring edits with a bounded length. Therefore, this dissertation focuses on correcting a substring edit of bounded length k with as low redundancy as possible, providing a universal error correction solution for a diverse set of localized errors.

1.2.3 Errors occurring in the enzymatic synthesis

Due to high cost[11], [22] and chemistry limitations [20], [45], DNA sequences produced by common synthesis methods have limited quantity and quality [38]. Recently, the authors in [38] proposed a new inexpensive enzymatic method, called *terminator-free template-independent* enzymatic DNA synthesis (simplified as enzymatic synthesis), to synthesize DNA sequences. The cost of synthesizing 1000 strands of 1000 nucleotide length by enzymatic synthesis is around one order of magnitude lower than the phosphoramidite technique [13], [33].

Different from conventional methods with single-base accuracy, in each synthesis round, a random number of nucleotides of the same type are appended to a sequence by the enzymatic synthesis. The random number satisfies a distribution that is affected by multiple factors, including previous bases and the *synthesis duration* of the current round [38]. After n synthesis rounds, this process produces N DNA sequences in parallel. The N synthesized DNA sequences have noisy lengths, as well as suffer deletions, insertions, and substitutions (of runs), where the deletions are dominant [38].

To combat those errors, the authors in [38] encode information in *transitions* between adjacent non-identical bases. Because the main cost stems from the time to synthesize the DNA molecules, we focus on the *writing rate*, defined as *bits per unit time*, in the system [28]. Since the run lengths are not considered, the writing rate of the work [38] is upper bounded by $\log_2 3$ bits per run[1]. The works in [28], [68] increase the writing rate by further making full use of run lengths of nucleotides in N traces without considering the deletions (of runs), the dominant errors. Therefore, this dissertation designs the error-correcting codes that can achieve a writing rate higher than $\log_2 3$ bits per unit time while fighting against deletions (of runs), the dominant errors in the enzymatic synthesis system.

1.3 Thesis outline, contributions, and notation

1.3.1 Thesis outline

The rest of the thesis is organized as follows. Chapter 2 and Chapter 3 present error-control codes to correct any duplications and edit errors, the first source of errors. More specifically, Chapter 2 constructs error-detecting codes and error-correcting codes for any fixed-length duplications and at most one *restricted or unrestricted* substitution, followed by the error-correcting codes to correct any short tandem duplications and at most p edit errors in Chapter 3. Chapter 4 focuses on constructing error-correcting codes to correct a substring edit of bounded length. Finally, the error-correcting codes to correct errors resulting from enzymatic synthesis are presented in Chapter 5.

1.3.2 Contributions

In order to provide reliable data in DNA storage, this thesis presents error-correcting codes to correct various sources of errors: i) fixed-length duplications and at most one substitution, ii) short

[1]We define the unit of time as the amount of time that it takes to produce a run in this method, so the writing rate in the work [38] is equivalent to $\log_2 3$ bits per unit time [28], [38].

duplications and at most p edits, iii) a substring edit error, and iv) errors in enzymatic synthesis. The main contributions are summarized below:

- Motivated by mutation processes occurring *in vivo* DNA-storage applications, Chapter 2 investigates two channel models, i.e., the *unrestricted substitution* and *restricted substitution (also called noisy duplication)* models, that mutate stored strings by fixed-length duplicating substrings as well as substituting symbols. error-detecting codes are constructed for the two models while error-correcting codes are constructed for *restricted substitution* channel, which can handle correctly an unlimited number of fixed-length tandem duplications and at most a single substitution occurring at any time during the mutation process. In particular, we show that the proposed code construction corrects a restricted substitution with the asymptotically optimal code rate as codes correcting exact duplications only.

- Apart from models with fixed-length duplications and at most one substitution, Chapter 3 extends works in [8], [30], [35] to correct an arbitrary number of short tandem duplications with duplication length upper bounded by 3 and at most p edits. This chapter starts by constructing MDS-based error-correcting codes to correct an arbitrary number of short duplications and at most one edit error with an additional rate penalty of 0.003 bits/symbol when the alphabet has size 4, an important case corresponding to data storage in DNA. Then by applying syndrome compression technique [75] and an extended MDS-based error-correcting code, the chapter further constructs low-redundancy error-correcting codes for simultaneously correcting short (tandem) duplications and at most p edits at the additional cost of roughly $8p(\log_q n)(1 + o(1))$ symbols of redundancy, thus achieving the same asymptotically optimal rate as codes correcting short (tandem) duplications only, where $q \geqslant 4$ is the alphabet size and p is a constant.

- Chapter 4 first shows through statistical analysis of real data that substring edits better describe differences between related documents compared to independent edits, and also shows that substring-edit-correcting codes can synchronize two documents with much lower overhead compared to general indel/substitution-correcting codes. Furthermore, given a constant k, this chapter presents binary codes of length n for correcting a single k-substring edit that achieves the GV bound and subsequently has redundancy of roughly $2 \log n$, outperforming the lowest redundancy $4k \log n$ achievable by an existing code for this problem. The time complexities of both encoding and decoding are polynomial with respect to n.

- For the newly proposed inexpensive enzymatic method, Chapter 5 proposes an error-correcting code and a decoding algorithm including sequence reconstruction and Bayes inference algorithms that can combat deletions and achieve a *writing rate* higher than $\log_2 3$ bits per unit time. In particular, for the specific *Poisson* distribution of run lengths with deletion errors, the numerical results showed that the code scheme can achieve a code rate of $\log_2 3$ bits per unit time and an error probability approximating the analytical results.

1.3.3 Notation

This subsection introduces some notation that will be used in the rest of the thesis. Note that more specific definitions or preliminaries appear in the corresponding chapters.

Without loss of generality, let $\Sigma_q = \{0, 1, 2, \cdots, q-1\}$ represent a finite alphabet of size q. Let Σ_q^n denote the set of all strings of length n over Σ_q. Let Σ_q^* denote the set of finite strings over Σ_q. Then we have $\Sigma_q^* = \bigcup_{n=0}^{\infty} \Sigma_q^n$. In particular, the empty string Λ is a member of the set Σ_q^*. Furthermore, let $\Sigma_q^+ = \Sigma_q^* \setminus \{\Lambda\}$. Given two integers a, b with $a \leqslant b$, the set of integers $\{a, a+1, \cdots, b\}$ is defined as $[a, b]$. We simplify $[a, b]$ as $[b]$ if $a = 1$. Let $\mathbb{N}$ and $\mathbb{N}_+$ denote the sets of nonnegative and positive integers, respectively. For an integer $a \geqslant 1$, let $b \bmod a$ be the remainder in $[0, a-1]$. Furthermore, we define $b \mod' a$ as the integer in $[a]$ whose remainder when divided by a is the same as that of b. Unless otherwise stated, logarithms are in base 2, i.e., $\log_2 5$ is denoted as $\log 5$.

In this thesis, let bold symbols denote strings over Σ_q, e.g., $\boldsymbol{x}, \boldsymbol{y}_j \in \Sigma_q^*$. The elements of a string are represented by plain typeface, e.g., the ith elements of $\boldsymbol{x}, \boldsymbol{y}_j \in \Sigma_q^*$ are $x_i, y_{ji} \in \Sigma_q$ respectively. For two strings $\boldsymbol{x}, \boldsymbol{y} \in \Sigma_q^*$, let $\boldsymbol{xy}$ or $(\boldsymbol{x}, \boldsymbol{y})$ denote their concatenation. Given four strings $\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w} \in \Sigma_q^*$, if $\boldsymbol{x}$ can be represented as $\boldsymbol{x} = \boldsymbol{uvw}$, then the string $\boldsymbol{v}$ is called a substring of $\boldsymbol{x}$. For a string $\boldsymbol{x}$, two substrings $\boldsymbol{u}$ and $\boldsymbol{v}$ are said to *overlap* if we can write $\boldsymbol{x} = \boldsymbol{abcde}$, where $\boldsymbol{c}$ is nonempty, $\boldsymbol{u} = \boldsymbol{bc}$, and $\boldsymbol{v} = \boldsymbol{cd}$. Furthermore, the length of a string $\boldsymbol{x} \in \Sigma_q^n$ is represented as $|\boldsymbol{x}|$. Given a set S, the size (the number of elements) of S is denoted as $\|S\|$.

Given a family of codes $\mathcal{C} = \{C_n\}_n$, where $C_n \subseteq \Sigma_q^n$, based on base 2 or q, the *code rate* is defined as

$$R_n(\mathcal{C}) = \frac{1}{n} \log_q \|C_n\| \text{ or } R_n(\mathcal{C}) = \frac{1}{n} \log \|\mathcal{C}_n\|, \tag{1.1}$$

where $\|C\|$ denotes the size of the code C. Furthermore, the *asymptotic rate* is defined as

$$R(\mathcal{C}) = \limsup_{n \to \infty} R_n(\mathcal{C}). \tag{1.2}$$

If the meaning is clear from the context, we may refer to both the family and individual codes as C and write $R_n(C), R(C)$ to refer to the rates for a given construction, where the unit is either bits/symbol (resp. symbols/symbol) for logarithms with base 2 (resp. q) respectively. Furthermore, the redundancy of the code is defined as $r(\mathcal{C}_n) = n - \log_q \|\mathcal{C}_n\|$.

Chapter 2

Detecting and correcting many k-duplications and one substitution

2.1 Introduction

Recall that, in Section 1.2.1, duplications [32], [36], [44], [49], [64] and edit errors [4], [38], [62], [78], [96] have been previously studied by a number of recent works independently. This chapter focuses on error-control codes for an arbitrary number of (tandem) duplications of fixed-length k, simplified as *k-TDs* or *k-duplications*, and at most one substitution error as all of them are observed in DNA data storage.

In a k-TD event, a substring of the DNA sequence of length-k, the *template*, is duplicated and the resulting *copy* is inserted into the sequence next to the template [103]. In a substitution event, a symbol in the sequence is changed to another symbol of the alphabet. It has been observed that point mutations such as substitutions are more common in tandem repeat regions of the genomes [62]. We consider two models for combined k-duplication and substitution errors. In the first model, called the *noisy-duplication model*, the copy is a noisy version of the template. Noisy duplications in this model can be viewed as exact k-duplications followed by substitutions that are restricted to the newly added copy. Hence, this model is also referred to as the *restricted-substitution model*, e.g., $ACGT \rightarrow ACGT\underline{CTT}$. We also consider an *unrestricted-substitution model*, which relaxes the noisy duplication model by allowing substitutions at any position in the sequence, e.g., $ACGT \rightarrow TCGT\underline{CGT}$.

The main approach in both cases is to reverse the k-duplication process while accounting for the single substitution (which may spuriously create the appearance of a duplication that never happened, or eliminate one that did). Different challenges are also presented by the possible locations for substitutions. We bring these differences to light by providing constructions for error-detecting codes and error-correcting codes for both the *restricted-substitution model* and the *unrestricted substitution model*.

Our main contributions are the following:

- We present an upper bound on the minimum required redundancy cost for detecting a single

restricted substitution, over the necessary rate loss required to correct an unlimited number of k-duplication events, in Theorem 12. That extra cost is upper bounded by $O(\log(n-k))$.

- Through Construction 17 and Construction 20, we also show that the redundancy cost (over the rate loss due to duplication noise) is upper bounded by $O(\log(k))$ and $O(k)$, respectively. While the former guarantees larger codes, it is nonconstructive, as opposed to the latter. In the likely regime where k is fixed, both require only $O(1)$ extra redundancy.

- Through Construction 26, we show that the redundancy cost of detecting a single unrestricted substitution (again, over the rate loss due to duplication noise) is upper bounded by $O\big(\log(k^2 n)\big)$.

- Finally, Construction 37 and Theorem 38 show that the extra redundancy cost of correcting a *noisy duplication* is roughly bounded by $(2k+4)\log_q n$, i.e., $(O(k\log(n)))$, without asymptotically rate loss.

This chapter is organized as follows. In Section 2.2, we provide the notation as well as relevant background and known results. In Section 2.3, we construct error-detecting codes for the restricted substitution model. In Section 2.4, we introduce error-detecting codes for unrestricted substitution channels. Finally, Section 2.5 constructs error-correcting codes to correct a restricted substitution (noisy duplication). We conclude with a discussion of the results, and point out some open problems, in section 2.6.

2.2 Notation and preliminaries

Throughout the chapter, we assume that the alphabet Σ is a unital ring of size $q \geqslant 2$ (e.g., $\mathbb{Z}_q$ or, when q is a prime power, $\mathbb{F}_q$). Thus, addition (or subtraction) and multiplications of letters from the alphabet are well-defined. The set of finite strings and strings of length at least k over Σ is denoted Σ^* and $\Sigma^{\geqslant k}$, respectively. The concatenation of two strings, $\boldsymbol{u}, \boldsymbol{v} \in \Sigma^*$ is denoted by $\boldsymbol{uv}$, and $\boldsymbol{u}^k$ denotes concatenating k copies of $\boldsymbol{u}$. To avoid confusion, the multiplication in the ring is denoted as $a \cdot b$. We say $\boldsymbol{y} \in \Sigma^*$ is a substring of $w \in \Sigma^*$ if there exists $\boldsymbol{x}, \boldsymbol{z} \in \Sigma^*$ such that $\boldsymbol{w} = \boldsymbol{xyz}$.

The length (number of letters) of $\boldsymbol{u}$ is denoted by $|\boldsymbol{u}|$, and for $a \in \Sigma$, we use $|\boldsymbol{u}|_a$ to denote the number of occurrences of a in $\boldsymbol{u}$. The Hamming weight of $\boldsymbol{u}$ is denoted by $\mathrm{wt}(\boldsymbol{u})$, and if $|\boldsymbol{u}| = |\boldsymbol{v}|$ we use $d(\boldsymbol{u}, \boldsymbol{v})$ to denote the Hamming distance between $\boldsymbol{u}$ and $\boldsymbol{v}$. If the need arises to refer to specific positions in words, positions are numbered $1, 2, \ldots$.

A (tandem) *duplication* of length k duplicates a substring of length k and inserts it in tandem into the string, namely, the copy immediately follows the template. More specifically, a k-TD can be expressed as [30]

$$T_{i,k}(\boldsymbol{x}) = \begin{cases} \boldsymbol{uvvw} & \text{if } \boldsymbol{x} = \boldsymbol{uvw}, |\boldsymbol{u}| = i, |\boldsymbol{v}| = k \\ \boldsymbol{x} & \text{if } |\boldsymbol{x}| < i + k \end{cases}$$

For example, from $\boldsymbol{uvw}$, where $|\boldsymbol{v}| = k$, we may obtain $\boldsymbol{uvvw}$. As an example for $k = 3$ and

alphabet $\Sigma = \mathbb{Z}_3$, consider

$$\boldsymbol{x} = 1012121 \to \boldsymbol{x}' = 1012\underline{012}121, \tag{2.1}$$

where the underlined part is the copy.

The analysis of k-duplication errors will be facilitated by the *k-discrete-derivative* transform, defined in [16] in the following way. For $\boldsymbol{x} \in \Sigma^{\geqslant k}$, we define $\phi(\boldsymbol{x}) \triangleq \hat{\phi}(\boldsymbol{x})\bar{\phi}(\boldsymbol{x})$, where

$$\hat{\phi}(\boldsymbol{x}) \triangleq x_1 \cdots x_k, \qquad \bar{\phi}(\boldsymbol{x}) \triangleq x_{k+1} \cdots x_n - x_1 \cdots x_{n-k}, \tag{2.2}$$

in which subtraction is performed entry-wise over Σ. We note that $\phi(\cdot)$ is a bijection. The duplication length k is implicit in the definition of ϕ. For a set of strings S, we define $\phi(S) \triangleq \{\phi(\boldsymbol{s}) \mid \boldsymbol{s} \in S\}$.

Let $\boldsymbol{x}'$ be obtained through a tandem duplication of length k from $\boldsymbol{x}$. It is not difficult to see that $\hat{\phi}(\boldsymbol{x}) = \hat{\phi}(\boldsymbol{x}')$ and that $\bar{\phi}(\boldsymbol{x}')$ can be obtained from $\bar{\phi}(\boldsymbol{x})$ by inserting 0^k in an appropriate position [32]. For the example given in (2.1),

$$\begin{aligned} \boldsymbol{x} = 1012121 &\to \boldsymbol{x}' = 1012\underline{012}121 \\ \phi(\boldsymbol{x}) = 101, 1112 &\to \phi(\boldsymbol{x}') = 101, 1\underline{000}112 \end{aligned} \tag{2.3}$$

Here, a comma separates the two parts of ϕ for clarity.

Sometimes duplications are *noisy* and the duplicated symbols are different from the original symbols. (Unless otherwise stated k-duplications are assumed to be exact.) We only consider the case where a single symbol is different, called *k-ND*. We view a noisy duplication as a k-duplication followed by a substitution in the duplicated substring. Continuing the example, the k-duplication resulting in $\boldsymbol{x}'$ may be followed by a substitution,

$$\begin{aligned} \boldsymbol{x}' = 1012\underline{0}12121 &\to \boldsymbol{x}'' = 1012\underline{1}12121, \\ \phi(\boldsymbol{x}') = 101, 1\underline{0}00\underline{1}12 &\to \phi(\boldsymbol{x}'') = 101, 1\underline{1}00\underline{0}12. \end{aligned}$$

We also consider unrestricted substitutions, which can occur at any position in the string, rather than only in a substring that is duplicated by the previous k-duplication. A substitution may be considered as the mapping $\boldsymbol{x} \to \boldsymbol{x} + a\boldsymbol{e}_i$, where $\boldsymbol{e}_i \in \Sigma^n$ is a standard unit vector at index i, and $a \in \Sigma$, $a \neq 0$. Since ϕ is linear over Σ (i.e., $\phi(\boldsymbol{x} + a\boldsymbol{e}_i) = \phi(\boldsymbol{x}) + a\phi(\boldsymbol{e}_i)$), we denote the transform of $\boldsymbol{e}_i$ as $\boldsymbol{\epsilon}_i \triangleq \phi(\boldsymbol{e}_i)$, and observe that $\epsilon_i = \boldsymbol{e}_i - \boldsymbol{e}_{i+k}$ for $i \leqslant n-k$ and $\boldsymbol{\epsilon}_i = \boldsymbol{e}_i$ for $n-k < i \leqslant n$. We note that substitutions might affect two positions in the ϕ-transform domain.

Let $D_k^{t(p)}(\boldsymbol{x})$ (for $t \geqslant p$) denote the set of strings that can be obtained from $\boldsymbol{x}$ through t tandem duplications of length k, p of which are noisy (in any order), with each noisy duplication containing a single substitution. $D_k^{t(p)}$ is called a *descendant cone* of $\boldsymbol{x}$. Continuing our earlier examples, we

have $\boldsymbol{x}' \in D_k^{1(0)}(\boldsymbol{x})$ and $\boldsymbol{x}'' \in D_k^{1(1)}(\boldsymbol{x})$. We further define

$$D_k^{*(p)}(\boldsymbol{x}) \triangleq \bigcup_{t=p}^{\infty} D_k^{t(p)}(\boldsymbol{x}), \quad D_k^{*(P)}(\boldsymbol{x}) \triangleq \bigcup_{p \in P} D_k^{*(p)}(\boldsymbol{x}), \tag{2.4}$$

where P is a subset of non-negative integers. We denote $P = \{0, 1\}$ as $\leqslant 1$.

We define $D_k^{t,p}(\boldsymbol{x})$ to be the set of strings obtained from $\boldsymbol{x}$ through t tandem duplications and p substitutions, where substitutions can occur in any position (and so we do not require $t \geqslant p$), and at any stage during the duplication sequence. We extend this definition similarly to (2.4). Obviously, for all $\boldsymbol{x} \in \Sigma^*$,

$$D_k^{t(0)}(\boldsymbol{x}) = D_k^{t,0}(\boldsymbol{x}).$$

For a string $\boldsymbol{z} \in \Sigma^*$, $\mu(\boldsymbol{z})$ is obtained by removing all copies of 0^k from $\boldsymbol{z}$. Specifically, for

$$\boldsymbol{z} = 0^{m_0} w_1 0^{m_1} w_2 \cdots w_d 0^{m_d},$$

where m_i are non-negative integers and $w_i \in \Sigma \setminus \{0\}$ are nonzero symbols, we define

$$\mu(\boldsymbol{z}) \triangleq 0^{m_0 \bmod k} w_1 0^{m_1 \bmod k} w_2 \cdots w_d 0^{m_d \bmod k},$$

where k is implicit in the notation $\mu(\boldsymbol{z})$. For example, if $\boldsymbol{z} = 1000112 = \bar{\phi}(\boldsymbol{x}')$ from our earlier example, with $k = 3$, then $\mu(\boldsymbol{z}) = 1112$; note, then, that in that example, $\mu(\boldsymbol{z}) = \bar{\phi}(\boldsymbol{x})$.

Define the *duplication root* $\mathrm{drt}(\boldsymbol{x})$ of $\boldsymbol{x}$ as the unique string obtained from $\boldsymbol{x}$ by removing all tandem repeats of length k, where the dependence on k is implicit in the notation. For proof of the uniqueness of $\mathrm{drt}(\boldsymbol{x})$ see, e.g., [32]. Note that

$$\phi(\mathrm{drt}(\boldsymbol{x})) = \hat{\phi}(\boldsymbol{x}) \mu(\bar{\phi}(\boldsymbol{x}))$$

(see [32]); indeed, in our running example, $\boldsymbol{x} = \mathrm{drt}(\boldsymbol{x}')$. For a set of strings S, we define

$$\mathrm{drt}(S) \triangleq \{\mathrm{drt}(\boldsymbol{s}) \mid \boldsymbol{s} \in S\}.$$

A string $\boldsymbol{x}$ is *k-irreducible* if $\boldsymbol{x} = \mathrm{drt}(\boldsymbol{x})$. The set of irreducible strings of length n is denoted $\mathrm{Irr}^{(\mathrm{k})}(n)$, where the duplication length k is again implicit. We denote by $\mathrm{RLL}(m)$ the set of strings in Σ^m that do not contain 0^k as a substring, i.e., the $(0, k-1)$-run-length limited (RLL) constrained strings of length m. In other words, $\mathrm{RLL}(m) = \{\boldsymbol{z} \in \Sigma_q^m | \mu(\boldsymbol{z}) = \boldsymbol{z}\}$. A string $\boldsymbol{x}$ of length n is irreducible if and only if $\bar{\phi}(\boldsymbol{x}) \in \mathrm{RLL}(n-k)$.

A code $C \subseteq \Sigma^n$ that can correct any number of k-duplication errors is called a *k-duplication code*. We note that a code is a k-duplication code if and only if no two distinct codewords $\boldsymbol{c}_1, \boldsymbol{c}_2 \in C$ have a common descendant, namely,

$$D_k^{*,0}(\boldsymbol{c}_1) \cap D_k^{*,0}(\boldsymbol{c}_2) = \varnothing.$$

It was proved in [32] that this condition is equivalent to all codewords having distinct roots:

Theorem 2. *([32]) For all strings, $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \Sigma^*$,*

$$D_k^{*,0}(\boldsymbol{x}_1) \cap D_k^{*,0}(\boldsymbol{x}_2) \neq \varnothing$$

if and only if $\mathrm{drt}(\boldsymbol{x}_1) = \mathrm{drt}(\boldsymbol{x}_2)$.

Using Theorem 2, it was suggested in [32] that error-correcting codes that protect against any number of k-duplications may be obtained simply by taking irreducible words as codewords. Up to a minor tweaking, this strategy was shown in [32] to produce optimal codes.

Finally, we recall the *redundancy* of a code $C \subseteq \Sigma^n$ as

$$r(C) \triangleq n - \log_q \|C\| = n - \log_{\|\Sigma\|} \|C\|,$$

and the code's *rate* (with logarithm in base q) as

$$R(C) \triangleq 1 - \frac{r(C)}{n}.$$

The key notations are summarized in Table 2.1

2.3 Restricted error-Detecting codes

2.3.1 The error model and the descendant cone

In this section, we consider the case of noisy-duplication errors. Our goal is to correct errors consisting of any number of exact k-duplications, or detect the presence of a single noisy duplication, which contains only one substitution. We refer to codes with this capability as *1-noisy duplication (1ND)* detecting. Let us first be more precise in our definition:

Definition 3. *A code $C \subseteq \Sigma^*$ is a 1ND-detecting code if there exists a decoding function $\mathcal{D} : \Sigma^* \to C \cup \{\text{error}\}$ such that if $\boldsymbol{c} \in C$ was transmitted and $\boldsymbol{y} \in \Sigma^*$ was received then $\mathcal{D}(\boldsymbol{y}) = \boldsymbol{c}$ if only k-duplication errors occurred, and $\mathcal{D}(\boldsymbol{y}) \in \{\boldsymbol{c}, \text{error}\}$ if exactly one of the k-duplication errors that occurred was noisy, where the noisy duplication could have occurred at any point in the sequence of the duplication errors.*

The following lemma, which relates the intersection of descendant cones to the intersection of the sets of roots of these cones, is of use in the discussion of 1ND-detecting codes.

Lemma 4. *For any strings $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \Sigma^*$ and sets $P_1, P_2 \subseteq \mathbb{Z}_{\geqslant 0}$,*

$$D_k^{*(P_1)}(\boldsymbol{x}_1) \cap D_k^{*(P_2)}(\boldsymbol{x}_2) \neq \varnothing$$

if and only if

$$\mathrm{drt}(D_k^{*(P_1)}(\boldsymbol{x}_1)) \cap \mathrm{drt}(D_k^{*(P_2)}(\boldsymbol{x}_2)) \neq \varnothing.$$

Table 2.1: Key notations in Chapter 2

Notation	Definition
$\Sigma_q = \{0, 1, \cdots, q-1\}$	The alphabet set with q elements
$\lvert\boldsymbol{u}\rvert$	The length of a sequence $\boldsymbol{u}$
$\lVert S\rVert$	The number of elements in the set S
$\phi(\boldsymbol{x}) = \hat{\phi}(\boldsymbol{x})\bar{\phi}(\boldsymbol{x})$	k-*discrete-derivative* transform
$\hat{\phi}(\boldsymbol{x})$	the first k elements $x_1 \cdots x_k$ of $\boldsymbol{x}$
$\bar{\phi}(\boldsymbol{x})$	the subtraction $x_{k+1} \cdots x_n - x_1 \cdots x_{n-k}$
$D_k^{t(p)}(\boldsymbol{x})$	noisy duplication descendant cone generated from $\boldsymbol{x}$ by t k-duplications (including p noisy duplications)
$D_k^{*(p)}(\boldsymbol{x})$	noisy duplication descendant cone generated from $\boldsymbol{x}$ by many k-duplications (including p noisy duplications)
$D_k^{t,p}(\boldsymbol{x})$	unrestricted duplication descendant cone generated from $\boldsymbol{x}$ by t k-duplications and p substitutions
$D_k^{*,p}(\boldsymbol{x})$	unrestricted duplication descendant cone generated from $\boldsymbol{x}$ by many k-duplications and p substitutions
$\mu(\boldsymbol{z})$	removing all substrings 0^k in $\boldsymbol{z}$
$\mathrm{drt}(\boldsymbol{x})$	duplication root of $\boldsymbol{x}$ by removing all tandem copies $\boldsymbol{a}$ of length k from $\boldsymbol{aa}$
$\mathrm{Irr}^{(\mathrm{k})}(n)$	the set of all length-n irreducible sequences without tandem repeats $\boldsymbol{aa}$ of length k
$\mathrm{RLL}(m)$	the set of length-m run-length-limited sequences without substrings 0^k
k-TD	a tandem duplication to generate a substring $\boldsymbol{aa}$ from $\boldsymbol{a}$
k-ND	a tandem duplication to generate a substring $\boldsymbol{ab}$ from $\boldsymbol{a}$, where $\boldsymbol{b}$ differs from $\boldsymbol{a}$ by one symbol

Proof. The 'only if' direction follows from definition. For the other direction, assume there exist $\boldsymbol{x}'_1 \in D_k^{*(P_1)}(\boldsymbol{x}_1)$ and $\boldsymbol{x}'_2 \in D_k^{*(P_2)}(\boldsymbol{x}_2)$ such that $\mathrm{drt}(\boldsymbol{x}'_1) = \mathrm{drt}(\boldsymbol{x}'_2)$. But then, by Theorem 2, there exists $\boldsymbol{x} \in D_k^{*(0)}(\boldsymbol{x}'_1) \cap D_k^{*(0)}(\boldsymbol{x}'_2)$. It follows that $\boldsymbol{x} \in D_k^{*(P_1)}(\boldsymbol{x}_1) \cap D_k^{*(P_2)}(\boldsymbol{x}_2)$. This is illustrated in Figure 2.1, where $\boldsymbol{y} = \mathrm{drt}(\boldsymbol{x}'_1) = \mathrm{drt}(\boldsymbol{x}'_2)$. □

We can now characterize 1ND-detecting codes in terms of duplication roots and descendant cones.

Lemma 5. *A code $C \subseteq \Sigma^n$ is a 1ND-detecting k-duplication code if and only if for any two distinct codewords $\boldsymbol{c}_1, \boldsymbol{c}_2 \in C$,*

$$D_k^{*(\leqslant 1)}(\boldsymbol{c}_1) \cap D_k^{*(0)}(\boldsymbol{c}_2) = \varnothing, \tag{2.5}$$

or equivalently,

$$\mathrm{drt}(\boldsymbol{c}_2) \neq \mathrm{drt}(\boldsymbol{c}_1), \tag{2.6}$$

$$\mathrm{drt}(\boldsymbol{c}_2) \notin \mathrm{drt}(D_k^{*(1)}(\boldsymbol{c}_1)). \tag{2.7}$$

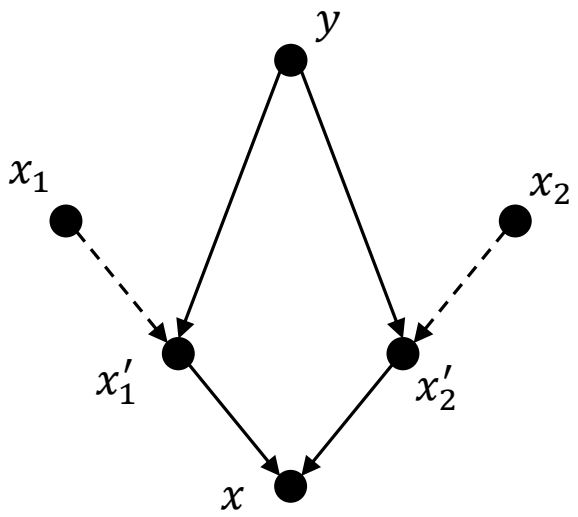


Figure 2.1: Illustration for the proof of Lemma 4. Solid lines denote any number of exact k-duplications and dashed lines represent a mixture of exact and noisy duplications (the number of noisy duplications is determined by P_1 and P_2).

Proof. Consider the following decoder: If there is a codeword with the same (exact-)duplication root as the received word, output that codeword. If not, declare that a noisy duplication error has occurred. Now, suppose (2.5) holds and that $\boldsymbol{c}_1$ is transmitted. If only exact k-duplications occur, the decoder outputs $\boldsymbol{c}_1$ since exact k-duplications do not alter the root and there is no other codeword $\boldsymbol{c}_2$ with the same root as $\boldsymbol{c}_1$. If, in addition, a noisy duplication occurs, then the received word either has the same root as $\boldsymbol{c}_1$ or it does not. Note again that the duplication root of the received word only changes as a result of the noisy duplication, regardless of when it occurs in the sequence of k-duplication events. In the former case, the decoder correctly outputs $\boldsymbol{c}_1$. In the latter case, (2.5) implies that no codeword has the same root as the received word, and thus the decoder correctly declares that a noisy duplication has occurred.

On the other hand, if (2.5) does not hold, no decoding method can both 'correct any number of exact k-duplications' and 'detect the presense of one noisy duplication'. That is because there exist distinct $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ and some $\boldsymbol{x} \in D_k^{*(\leqslant 1)}(\boldsymbol{c}_1) \cap D_k^{*(0)}(\boldsymbol{c}_2)$. If $\boldsymbol{x}$ is received then there is no way to determine whether $\boldsymbol{c}_1$ or $\boldsymbol{c}_2$ was transmitted.

The equivalence between (2.5) and (2.6, 2.7) follows from Lemma 4. □

Based on Lemma 5, we consider codes whose distinct codewords satisfy (2.6) and (2.7). Further, the decoder outputs the codeword with the same root as the retrieved word if it exists, and otherwise declares a noisy duplication.

As a result of the substitution in the noisy duplication error, the length of the duplication root may change. One way to simplify the code design is to restrict ourselves to codes whose codewords all have duplication roots with the same length. Then, error patterns that modify this length can be easily detected and we can focus on patterns that keep the duplication-root length the same. Specifically, for a given length n, we consider codes whose codewords are irreducible strings of length n. The effect of this restriction on the size of the code is discussed following Theorem 12.

Definition 6. *A substitution error (as a component of a noisy-duplication error) that changes the root but not the length of the root is called an* ambiguous substitution.

It is easy to verify that when $k = 1$ a noisy duplication is never ambiguous. Thus, challenges arise only when $k \geqslant 2$. The following sequence of lemmas characterize the conditions under which a substitution is ambiguous.

Lemma 7. *Let $\boldsymbol{x} \in \Sigma^*$ be some string resulting from a k-duplication, $k \geqslant 2$. If a substitution occurs (as part of a noisy duplication) in the last k positions of $\boldsymbol{x}$ then it is not ambiguous.*

Proof. Since a substitution that occurs as part of a noisy duplication changes the copied part, we must have $\boldsymbol{z} \triangleq \bar{\phi}(\boldsymbol{x}) = \boldsymbol{u}0^k\boldsymbol{w}$, with $|\boldsymbol{w}| \leqslant k-1$. After the substitution we get $\boldsymbol{x}'$, with $\boldsymbol{z}' \triangleq \bar{\phi}(\boldsymbol{x}') = \boldsymbol{u}0^{k-i-1}b0^i\boldsymbol{w}$, for some $b \in \Sigma \setminus \{0\}$ and $i + |\boldsymbol{w}| \leqslant k-1$. It is, however, obvious that $|\mu(\boldsymbol{z})| < |\mu(\boldsymbol{z}')|$, and thus $|\mathrm{drt}(\boldsymbol{x})| < |\mathrm{drt}(\boldsymbol{x}')|$. □

Lemma 8. *Let $\boldsymbol{x} \in \Sigma^*$ be some string resulting from a k-duplication, $k \geqslant 2$. If $\boldsymbol{x}'$ is obtained from $\boldsymbol{x}$ as a result of a substitution that occurs (as part of a noisy duplication) in position $\ell \leqslant |\boldsymbol{x}| - k$, and in $\phi(\boldsymbol{x})$ positions $\ell+1, \ldots, \ell+k-1$ contain only zeros, then the substitution is not ambiguous.*

Proof. Denote $z = \bar{\phi}(\boldsymbol{x})$. Assume $\boldsymbol{z}' \triangleq \boldsymbol{z} + b \cdot \boldsymbol{\epsilon}_{\ell-k}$ (where the subscript is indeed $\ell - k$ since by considering $\bar{\phi}(\boldsymbol{x})$ we are omitting the prefix $\hat{\phi}(\boldsymbol{x})$ of length k). Then we may write

$$\begin{array}{rcccccc} \boldsymbol{z} & = & \boldsymbol{u} & 0 & 0^{k-1} & b' & \boldsymbol{w} \\ b \cdot \boldsymbol{\epsilon}_{\ell-k} & = & 0^{|\boldsymbol{u}|} & b & 0^{k-1} & (-b) & 0^{|\boldsymbol{w}|} \\ \boldsymbol{z}' & = & \boldsymbol{u} & b & 0^{k-1} & (b'-b) & \boldsymbol{w} \end{array}$$

where $\boldsymbol{u} \in \Sigma^{\ell-k-1}$, $\boldsymbol{w} \in \Sigma^*$, $b \in \Sigma \setminus \{0\}$, and $b' \in \Sigma$. We now have two cases. If $b' \neq b$, then obviously $|\mu(\boldsymbol{z})| < |\mu(\boldsymbol{z}')|$, namely $|\mathrm{drt}(\boldsymbol{x})| < |\mathrm{drt}(\boldsymbol{x}')|$. If $b' = b$, then $\mathrm{drt}(\boldsymbol{x}) = \mathrm{drt}(\boldsymbol{x}')$, which is again not ambiguous. □

The remaining cases are all handled in the following lemma.

Lemma 9. *Let $\boldsymbol{x} \in \Sigma^*$ be some string resulting from a k-duplication, $k \geqslant 2$, and let $\boldsymbol{x}'$ be obtained from x as a result of a substitution that occurs as part of a noisy duplication. Denote $\boldsymbol{z} \triangleq \bar{\phi}(\boldsymbol{x})$ and $\boldsymbol{z}' \triangleq \bar{\phi}(\boldsymbol{x}') = \boldsymbol{z} + \boldsymbol{\epsilon}_{\ell-k}$. Assume*

$$\begin{array}{rcccccccc} \boldsymbol{z} & = & \boldsymbol{u} & 0^{pk+m+i-1} & 0 & 0^{k-i} & \boldsymbol{v} & b' & \boldsymbol{w} \\ b \cdot \epsilon_{\ell-k} & = & 0^{|\boldsymbol{u}|} & 0^{pk+m+i-1} & b & 0^{k-i} & 0^{|\boldsymbol{v}|} & (-b) & 0^{|\boldsymbol{w}|} \\ \boldsymbol{z}' & = & \boldsymbol{u} & 0^{pk+m+i-1} & b & 0^{k-i} & \boldsymbol{v} & (b'-b) & \boldsymbol{w} \end{array}$$

where $\boldsymbol{u}, \boldsymbol{w} \in \Sigma^*$, $\boldsymbol{v} \in \Sigma^{i-1}$, $\boldsymbol{v}$ is not empty and begins with a non-zero letter, $b \in \Sigma \setminus \{0\}$, $b' \in \Sigma$, the run of zeros 0^{pk+m+k} in $\boldsymbol{z}$ between $\boldsymbol{u}$ and $\boldsymbol{v}$ is maximal, $p \in \mathbb{Z}_{\geqslant 0}$, $0 \leqslant m < k$, $1 < i \leqslant k$. Furthermore, denote the length of the run of zeros to the left of b' in $\boldsymbol{z}$ by m_1, and to its right by m_2. Then the substitution is ambiguous exactly when either:

C.1 $1 < i \leqslant k - m$, $b' = b$, *and* $\lfloor \frac{m_2}{k} \rfloor < \lfloor \frac{m_1+m_2+1}{k} \rfloor$.

Table 2.2: Examples of ambiguous substitution errors found in Lemma 9 over Σ_3. In all cases $\boldsymbol{y} = \hat{\phi}(\boldsymbol{x})$, $\boldsymbol{z} = \bar{\phi}(\boldsymbol{x})$, $\boldsymbol{z}' = \bar{\phi}(\boldsymbol{x}')$

1.1	2.3
$\boldsymbol{x} = 12122022002200$ $(\boldsymbol{y}, \boldsymbol{z}) = (121, 102\underline{00}010201)$ $\mathrm{drt}(\boldsymbol{x}) = 12122002200$	$\boldsymbol{x} = 12122122002200$ $(\boldsymbol{y}, \boldsymbol{z}) = (121, 10\underline{000}210201)$ $\mathrm{drt}(\boldsymbol{x}) = 12122002200$
$\boldsymbol{x}' = 12122022\underline{0}2200$ $(\boldsymbol{y}, \boldsymbol{z}') = (121, 10200\underline{2}10\underline{0}01)$ $\mathrm{drt}(\boldsymbol{x}') = 12122\underline{0}22200$	$\boldsymbol{x}' = 12122120002200$ $(\boldsymbol{y}, \boldsymbol{z}') = (121, 1000\underline{1}21\underline{2}201)$ $\mathrm{drt}(\boldsymbol{x}') = 1212\underline{0}002200$

C.2 $k - m < i \leqslant k$ *and* ($b' \notin \{0, b\}$ *or* $\lfloor \frac{m_2}{k} \rfloor = \lfloor \frac{m_1+m_2+1}{k} \rfloor$).

Proof. The following cases are possible:

1. If $1 < i \leqslant (k - m)$ then:

   1.1 if $b' = b$ and $\lfloor \frac{m_2}{k} \rfloor < \lfloor \frac{m_1+m_2+1}{k} \rfloor$, then a run of 0s of length at least k will be created in $\boldsymbol{z}'$, leading to $|\mu(\boldsymbol{z}')| = |\mu(\boldsymbol{z})|$ but $\mu(\boldsymbol{z}') \neq \mu(\boldsymbol{z})$. Thus the substitution is ambiguous.

   1.2 if $b' = 0$ and $\lfloor \frac{m_2}{k} \rfloor < \lfloor \frac{m_1+m_2+1}{k} \rfloor$, then length of the root over all increases by $2k$.

   1.3 in all other cases, the root's length increases by k.

2. If $(k - m) < i \leqslant k$, then a run of 0s of length $m + i - 1 \geqslant k$ will exist before b, implying that the length of the root before $\boldsymbol{v}$ will not change. Then:

   2.1 if $b' = b$ and $\lfloor \frac{m_2}{k} \rfloor < \lfloor \frac{m_1+m_2+1}{k} \rfloor$, then the length of the root decreases by k.

   2.2 if $b' = 0$ and $\lfloor \frac{m_2}{k} \rfloor < \lfloor \frac{m_1+m_2+1}{k} \rfloor$, then the length of the root increases by k.

   2.3 in all other cases, the length of the root remains the same, resulting in an ambiguous substitution.

□

Examples for the two cases in which ambiguous substitutions occur, as described in Lemma 9, are given in Table 2.2.

2.3.2 Bounds on the size of the code

We use the analysis of the previous section to find lower bounds on the size of 1ND-detecting codes. For $x \in \Sigma^n$, a quantity that will be useful in bounding the size of codes is the following:

$$V(\boldsymbol{x}) \triangleq \| \mathrm{drt}(D_k^{*(\leqslant 1)}(\boldsymbol{x})) \cap \Sigma^n \|.$$

This counts the number of strings $\boldsymbol{x}'$ that can be obtained from $\boldsymbol{x}$ through any number of k-duplications, at most one of them noisy, and such that $|\mathrm{drt}(\boldsymbol{x})| = |\mathrm{drt}(\boldsymbol{x}')|$.

Lemma 10. *For $x \in \mathrm{Irr}^{(\mathrm{k})}(n)$, where $n \geqslant 2k \geqslant 4$,*

$$V(x) \leqslant (n-k)(q-1) - \mathrm{wt}\big(\bar{\phi}(\boldsymbol{x})\big)(q-2).$$

Proof. We first assume, without loss of generality, that the noisy duplication occurs last, since subsequent k-duplications (which are not noisy) do not change the duplication root. Assume the notation is as defined in Lemma 9.

We first bound the contribution of the case 1.1 of the proof of Lemma 9 to $V(\boldsymbol{x})$. Since $n \geqslant 2k$ and $\boldsymbol{x}$ is irreducible, we have that $\mathrm{wt}(\boldsymbol{z}) \geqslant 1$. There are $\mathrm{wt}(\boldsymbol{z})$ non-zero elements in $\boldsymbol{z}$ that can serve as the first letter of $\boldsymbol{v}$, which we shall call the *anchor*. In this case, $b' \neq 0$, and it is found at most $k-m-1$ positions after the anchor. We contend that there is at most one such choice for b'. Indeed, if we are in case 1.1, then there is a run of m_1 zeros immediately to the left of b', and m_2 to the right. But

$$\left\lfloor \frac{m_1+m_2+1}{k} \right\rfloor > \left\lfloor \frac{m_2}{k} \right\rfloor \geqslant 0,$$

implying

$$m_1 + m_2 + 1 \geqslant k.$$

Thus, if case 1.1 holds then there is a single non-zero element in the k positions following the anchor. Additionally, since $b' = b$, we have a single choice for the value of b. Finally, we note that case 1.1 cannot occur when the anchor is the last non-zero element in $\boldsymbol{z}$. Hence, in total, the contribution of case 1.1 does not exceed $\mathrm{wt}(\boldsymbol{z}) - 1$.

We now turn to the case of 2.3. Assuming an anchor was chosen, the value of i can take at most m values, which is the length of the run of zeros before the anchor, taken modulo k. Ranging over all the run's zeros, the effect of modulo k simply leaves us with a choice of a position containing a 0 in $\boldsymbol{z}$, since $\boldsymbol{x}$ is irreducible. There are $n-k-\mathrm{wt}(\boldsymbol{z})$ such positions. Finally, there are at most $q-1$ possibilities for b. Thus, this case contributes at most $(n-k-\mathrm{wt}(\boldsymbol{z}))(q-1)$ to $V(\boldsymbol{x})$. Noting that $\boldsymbol{x}$ itself also contributes to $V(x)$ completes the proof. □

To find a lower bound on the size of the code, we apply the Gilbert-Varshamov (GV) bound with the average size of the sphere (see, e.g., [92]).

Lemma 11. *Let $\boldsymbol{x}$ be a randomly and uniformly chosen string from $\mathrm{Irr}^{(\mathrm{k})}(n)$. If $n \geqslant 2k \geqslant 4$, then*

$$\mathbb{E}[V(\boldsymbol{x})] \leqslant 2(n-k)(q-1)/q.$$

Proof. Let $\boldsymbol{z} = \bar{\phi}(\boldsymbol{x})$. From Lemma 10, to find the expected value of $V(\boldsymbol{x})$, it suffices to find the expected value of $\mathrm{wt}(\boldsymbol{z})$.

Fix i and let U be the set of strings obtained by removing position i from the strings in $\mathrm{RLL}(n-k)$ (if multiple copies of a string exist we keep only one). Let S be the set of strings $\boldsymbol{s}$ in U that contain a run of 0s of length at least $k-1$ that includes s_{i-1} or s_i. Furthermore, let $S^c = U \setminus S$. Now, the number of strings in $\mathrm{RLL}(n-k)$ that contain a 0 in position i equals $\|S^c\|$, while the total number

of strings in $\mathrm{RLL}(n-k)$ equals $\|S^c\|q + \|S\|(q-1)$. Hence, for a randomly chosen $\boldsymbol{z} \in \mathrm{RLL}(n-k)$,

$$\Pr(z_i = 0) = \frac{\|S^c\|}{\|S^c\|q + \|S\|(q-1)} \leqslant \frac{1}{q}$$

Thus, $\mathbb{E}[\mathrm{wt}(\boldsymbol{z})] \geqslant (n-k)(q-1)/q$. The result then follows from Lemma 10. □

The above lemma leads to the lower bound in the following theorem.

Theorem 12. *For positive integers $n \geqslant 2k \geqslant 4$, the maximum size $A_{1\mathrm{ND}}(n, q, k)$ of a 1ND-detecting codes of length n over $\mathbb{Z}_q$ satisfies*

$$\frac{1}{4(n-k)} \cdot M \leqslant A_{1\mathrm{ND}}(n, q, k) \leqslant M,$$

where

$$M \triangleq \sum_{i=0}^{\lfloor n/k \rfloor - 1} \|\mathrm{Irr}^{(\mathrm{k})}(n-ik)\| = \sum_{i=1}^{\lfloor n/k \rfloor} q^k \|\mathrm{RLL}(n-ik)\| \tag{2.8}$$

is the number of irreducible words whose descendant cones intersect Σ^n.

Proof. First we show that

$$\frac{q^{k+1}\|\mathrm{RLL}(n-k)\|}{2(n-k)(q-1)} \leqslant A_{1\mathrm{ND}}(n, q, k) \leqslant M.$$

The lower bound follows by applying the generalized GV bound [92] with Lemma 11. The upper bound follows from the fact that the code must be able to correct any number of k-duplication errors and from [32] where such codes are discussed.

To get the lower bound to the more appealing form we claim, we note that to any string of length $m-k$ that has no 0^k substring, we can append a string of length k whose first element is nonzero, and thus obtain a string of length m that has no 0^k substring. Hence,

$$\|\mathrm{RLL}(m)\| \geqslant \|\mathrm{RLL}(m-k)\|(q-1)q^{k-1}.$$

Thus

$$\|\mathrm{RLL}(n-ik)\| \leqslant \frac{\|\mathrm{RLL}(n-k)\|}{(q-1)^{i-1}q^{(i-1)(k-1)}}.$$

We then have

$$\begin{aligned} M &= \sum_{i=1}^{\lfloor n/k \rfloor} q^k \|\mathrm{RLL}(n-ik)\| \\ &\leqslant q^k \|\mathrm{RLL}(n-k)\| \sum_{i=1}^{\lfloor n/k \rfloor} \frac{1}{(q-1)^{i-1}q^{(i-1)(k-1)}} \\ &\leqslant q^k \|\mathrm{RLL}(n-k)\| \sum_{i=1}^{\infty} \frac{1}{(q-1)^{i-1}q^{(i-1)(k-1)}} \end{aligned}$$

$$\leqslant q^k \|\mathrm{RLL}(n-k)\| \frac{(q-1)q^{k-1}}{(q-1)q^{k-1}-1}.$$

Since $q+k \geqslant 4$ with $q, k \geqslant 2$,

$$\|\mathrm{Irr}^{(\mathrm{k})}(n)\| = q^k \|\mathrm{RLL}(n-k)\| \geqslant M/2, \tag{2.9}$$

and we have the desired claim. □

2.3.3 Code construction

The goal of this section is to construct 1ND-detecting codes. We shall first consider an auxiliary code construction which will be useful not only here, but also in the following section. The error we would like to detect by this auxiliary code is as follows:

Definition 13. *For $n, k > 0$, let $\boldsymbol{z}, \boldsymbol{z}' \in \Sigma^n$ be some strings. If we can write*

$$\begin{aligned} \boldsymbol{z} &= \boldsymbol{u} \quad \boldsymbol{v} \quad \boldsymbol{w}\ 0^{|\boldsymbol{v}|}\ x \\ \boldsymbol{z}' &= \boldsymbol{u}\ 0^{|\boldsymbol{v}|}\ \boldsymbol{w} \quad \boldsymbol{v} \quad \boldsymbol{x} \end{aligned}$$

where $\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}, \boldsymbol{x} \in \Sigma^*$, $1 \leqslant |\boldsymbol{v}| \leqslant k-1$, $\boldsymbol{v}$ is a non-zero string, and $|\boldsymbol{v}| + |\boldsymbol{w}| = k$, then we say $\boldsymbol{z}$ and $\boldsymbol{z}'$ differ by a single k-switch error.

Intuitively, a single k-switch error takes a non-zero non-empty substring of length at most $k-1$, and switches it with an all-zero substring of the same length found k positions before or after it.

Any non-empty string $\boldsymbol{z} \in \Sigma^n$ may be partitioned into non-overlapping blocks of length k:

$$z = B_1(\boldsymbol{z}) B_2(\boldsymbol{z}) \dots B_{\lceil n/k \rceil}(\boldsymbol{z}),$$

where $B_i(\boldsymbol{z}) \in \Sigma^k$ for all i, except if k does not divide n, in which case, $B_{\lceil n/k \rceil}(\boldsymbol{z}) \in \Sigma^{n \bmod k}$. We note that k is implicit in the definition of $B_i(\boldsymbol{z})$.

We now give a construction for a family of codes which we then show are all capable of detecting a single k-switch error.

Construction 14. *Let $k \geqslant 2$ and let p be the smallest odd integer larger than $k-1$, namely*

$$p \triangleq 2\left\lceil \frac{k-1}{2} \right\rceil + 1.$$

Fix a code length $n \in \mathbb{N}$ and let $S \subseteq \Sigma^n$ be an arbitrary set of strings. For any string $\boldsymbol{x} \in S$, and $\ell = 0, 1, 2, 3$, we define

$$Z_\ell(\boldsymbol{x}) \triangleq \sum_{i \in I_\ell} |B_i(\boldsymbol{x})|_0,$$

where $I_\ell = \{1 \leqslant t \leqslant \lceil n/k \rceil \mid t \equiv \ell \pmod 4\}$. For all $0 \leqslant i, j < p$, we construct

$$C^{\mathrm{aux}}_{i,j}(S) \triangleq \left\{ x \in S \;\middle|\; Z_0(\boldsymbol{x}) + 2Z_2(\boldsymbol{x}) \equiv i \pmod p \right.,$$

$$Z_1(\boldsymbol{x}) + 2Z_3(\boldsymbol{x}) \equiv j \pmod{p}\big\}.$$

Theorem 15. *Each code $C_{i,j}^{\mathrm{aux}}(S)$ of Construction 14 can detect a single k-switch error or a single zero replaced by a non-zero letter.*

Proof. Since $k \geqslant 2$ we have $p \geqslant 3$ which immediately enables the detection of a single zero replaced by a non-zero letter. Let us therefore focus on the problem of detecting a single k-switch error.

We assume $n \geqslant k+1$, otherwise the claim is trivial. Assume $\boldsymbol{x} \in C_{i,j}^{\mathrm{aux}}(S)$ sustains a single k-switch error, resulting in the string $\boldsymbol{x}' \in \Sigma^n$. For $0 \leqslant \ell \leqslant 3$, let

$$\Delta_\ell \triangleq Z_\ell(\boldsymbol{x}') - Z_\ell(\boldsymbol{x}).$$

Furthermore, for $0 \leqslant \ell \leqslant 1$, let

$$F_\ell \triangleq \Delta_\ell + 2\Delta_{\ell+2}.$$

To prove the error detection capabilities of the code it now suffices to show that

$$F_0 \not\equiv 0 \pmod{p} \qquad \text{or} \qquad F_1 \not\equiv 0 \pmod{p}. \tag{2.10}$$

Based on the definition of a k-switch error, the number of zeros changes in some blocks. We consider the following possible cases.

First, if the number of zeros changes in 2 consecutive blocks, then one of the pairs (Δ_0, Δ_1), (Δ_1, Δ_2), (Δ_2, Δ_3), (Δ_3, Δ_0) equals $(\delta, -\delta)$ for $0 < |\delta| < k$, and the two other Δ's are equal to 0. Then, $|F_0| = |\delta|$ or $|F_0| = 2|\delta|$. In the former case $F_0 \not\equiv 0 \pmod{p}$ since $0 < |\delta| < k \leqslant p$. In the latter case, $F_0 \not\equiv 0 \pmod{p}$ since $0 < 2|\delta| < 2p$ and $2\delta \neq p$ (recall that p is odd).

Second, if the number of zeros changes in two non-consecutive blocks, then only one of the pairs (Δ_0, Δ_2) and (Δ_1, Δ_3) equals $(\delta, -\delta)$ for $0 < |\delta| < k$, and the other equals $(0, 0)$. Then, either $|F_0| = |\delta|$ or $|F_1| = |\delta|$, and in both cases (2.10) is satisfied.

Third, if the change of number of zeros occurs in three consecutive blocks, then there exists ℓ such that $\Delta_\ell = \delta' \neq 0$ and $\Delta_{2+\ell} = 0$ (indices taken modulo 4), where $0 < |\delta'| < k$ and $2|\delta'| \neq p$. Then either F_0 or F_1 takes on the value of δ' or $2\delta'$. But $\delta' \not\equiv 0 \pmod{p}$ and $2\delta' \not\equiv 0 \pmod{p}$, implying that (2.10) is satisfied. □

We now turn to construct 1ND-detecting codes. As before, we consider codes that consist of irreducible strings of length n. We thus need to devise a method to detect ambiguous substitutions.

As mentioned before, when $k = 1$ ambiguous substitutions cannot occur. Hence $\mathrm{Irr}^{(\mathrm{k})}(n)$ is a 1ND-detecting code. For $k \geqslant 2$, our analysis rests on the following lemma.

Lemma 16. *Let $k \geqslant 2$. If $\boldsymbol{x} \in \Sigma^*$ and $\boldsymbol{x}'$ is obtained from $\boldsymbol{x}$ via any number of k-duplications among which one contains an ambiguous substitution, then $\bar{\phi}(\mathrm{drt}(\boldsymbol{x}))$ and $\bar{\phi}(\mathrm{drt}(\boldsymbol{x}'))$ differ by a single k-switch error, or*

$$\big|\big|\bar{\phi}(\mathrm{drt}(\boldsymbol{x}))\big|_0 - \big|\bar{\phi}(\mathrm{drt}(\boldsymbol{x}'))\big|_0\big| = 1.$$

Proof. Denote $\boldsymbol{z} \triangleq \bar{\phi}(\boldsymbol{x})$ and $\boldsymbol{z}' \triangleq \bar{\phi}(\boldsymbol{x}')$. With the notation of Lemma 9, one can verify that in Case 1.1 we have

$$\begin{aligned} \mu(\boldsymbol{z}) &= \boldsymbol{u}' \;\; \boldsymbol{v} \;\; 0^{i-1-|\boldsymbol{v}|}b0^{k-i} \; 0^{|\boldsymbol{v}|} \; \boldsymbol{w}' \\ \mu(\boldsymbol{z}') &= \boldsymbol{u}' \; 0^{|\boldsymbol{v}|} \; 0^{i-1-|\boldsymbol{v}|}b0^{k-i} \;\; \boldsymbol{v} \;\; \boldsymbol{w}' \end{aligned} \tag{2.11}$$

and in Case 2.3,

$$\begin{aligned} \mu(\boldsymbol{z}) &= \boldsymbol{u}' \; 0 \; 0^{k-|\boldsymbol{v}|-1}\boldsymbol{v} \quad\;\; b' \quad\;\; \boldsymbol{w}' \\ \mu(\boldsymbol{z}') &= \boldsymbol{u}' \; b \; 0^{k-|\boldsymbol{v}|-1}\boldsymbol{v} \; (b'-b) \; \boldsymbol{w}' \end{aligned} \tag{2.12}$$

for some $\boldsymbol{u}', \boldsymbol{w}' \in \Sigma^*$. In (2.11) we see a single k-switch error. In (2.12), if $b' = b$ we have a single k-switch error, and if $b \neq b'$ then the number of zeros differ by one. □

Construction 17. *Let n, k be positive integers, $n \geqslant k$, and let $S \triangleq \mathrm{RLL}(n-k)$. For all $0 \leqslant i, j < p$, we construct*

$$C_{i,j} \triangleq \{\phi^{-1}(\boldsymbol{y}\boldsymbol{z}) \;\mid\; \boldsymbol{y} \in \Sigma^k, \boldsymbol{z} \in C_{i,j}^{\mathrm{aux}}(S)\},$$

where p and $C_{i,j}^{\mathrm{aux}}(S)$ are defined in Construction 14.

Theorem 18. *With the setting as in Construction 17, the code $C_{i,j}$ is a 1ND-detecting code.*

Proof. By our choice of S, we necessarily have that $C_{i,j} \subseteq \mathrm{Irr}^{(\mathrm{k})}(n)$. If $k = 1$, then $C_{0,0} = \mathrm{Irr}^{(\mathrm{k})}(n)$ is the only code and the theorem is immediate.

Assume $k \geqslant 2$. Let $\boldsymbol{c}_1, \boldsymbol{c}_2 \in C_{i,j}$ be distinct codewords. Since $C_{i,j} \subseteq \mathrm{Irr}^{(\mathrm{k})}(n)$, $\mathrm{drt}(\boldsymbol{c}_1) = \boldsymbol{c}_1$ and $\mathrm{drt}(\boldsymbol{c}_2) = \boldsymbol{c}_2$, which are distinct. Based on (2.7) it suffices to show that for any $\boldsymbol{c}_1' \in D_k^{*(1)}(\boldsymbol{c}_1)$, we have $\boldsymbol{c}_2 \neq \mathrm{drt}(\boldsymbol{c}_1')$.

If $\mathrm{drt}(\boldsymbol{c}_1') = \mathrm{drt}(\boldsymbol{c}_1) = \boldsymbol{c}_1$, then clearly $\boldsymbol{c}_2 \neq \mathrm{drt}(\boldsymbol{c}_1')$. So we assume $\mathrm{drt}(\boldsymbol{c}_1') \neq \boldsymbol{c}_1$. It is then sufficient to show that $\mathrm{drt}(\boldsymbol{c}_1') \notin C_{i,j}$. This is obvious if $|\mathrm{drt}(\boldsymbol{c}_1')| \neq n$ and the substitution is not ambiguous. If the substitution is ambiguous, we obtain the claimed result by combining Lemma 16 and Theorem 15. □

Corollary 19. *If $n \geqslant k \geqslant 2$ then*

$$A_{1\mathrm{ND}}(n, q, k) \geqslant \frac{1}{2(k+1)^2} \cdot M,$$

where M is given by (2.8).

Proof. Let p and $C_{i,j}$ be defined as in Construction 17. The set $\{C_{i,j} \;\mid\; 0 \leqslant i, j < p\}$ forms a partition of $\mathrm{Irr}^{(\mathrm{k})}(n)$. Thus, a simple averaging argument shows that there exist i and j such that

$$\|C_{i,j}\| \geqslant \frac{\|\mathrm{Irr}^{(\mathrm{k})}(n)\|}{p^2}.$$

Since $p \leqslant k+1$, and by (2.9), we obtain the claim. □

Note that the lower bound on $A_{1\mathrm{ND}}(n, q, k)$ in this corollary may be better than the one given in Theorem 12.

The problem with the bound of Corollary 19 is that it is not constructive. In particular, we do not know exactly what choice of i and j gives the largest code $C_{i,j}$ in Construction 17. Construction 20 below provides a sub-code of $C_{0,0}$ from Construction 17 whose size can be lower bounded, albeit, somewhat smaller than the guarantee of Corollary 19.

Construction 20. *Let $k \geqslant 2$ and let p be the smallest odd integer larger than $k-1$, namely*

$$p \triangleq 2\left\lceil \frac{k-1}{2} \right\rceil + 1.$$

Fix a code length $n \in \mathbb{N}$, $n \geqslant 5k$. We construct a code $C \subseteq \Sigma^n$ in the following way: For each $y \in \mathrm{RLL}(n-5k)$, construct four strings of length k, denoted $B_0, B_1, B_2, B_3 \in \Sigma^k$,

$$B_i = 0^{\beta_i} 1^{k-\beta_i}, \qquad \text{for all } 0 \leqslant i \leqslant 3$$

where

$$\begin{aligned} \beta_i &= (-(\zeta_i + 2\zeta_{i+2}) \bmod p) - 2\beta_{i+2}, && i = 0, 1 \\ \beta_{i+2} &= \left\lfloor \frac{(-(\zeta_i + 2\zeta_{i+2}) \bmod p)}{2} \right\rfloor, && i = 0, 1 \\ \zeta_i &= Z_i(\phi^{-1}(0^k y)), && i = 0, 1, 2, 3 \end{aligned}$$

and add the codewords $\phi^{-1}(BB_0B_1B_2B_3y)$ where B runs over all strings in Σ^k.

Theorem 21. *Let q be the alphabet size, k the duplication length, $q + k \geqslant 4$, and $n \in \mathbb{N}$, $n \geqslant 5k$. Then the code C from Construction 20 is a 1ND-detecting code of size*

$$\|C\| = \|\operatorname{Irr}^{(\mathrm{k})}(n-4k)\| \geqslant \frac{1}{2 \cdot q^{4k}} \cdot M,$$

where M is given in (2.8).

Proof. One can easily verify that $0 \leqslant \beta_i < k$, hence all the blocks B_i end with a non-zero symbol and therefore all the codewords are irreducible. Additionally, by inspection we can verify that $C \subseteq C_{0,0}$, where $C_{0,0}$ is obtained from Construction 17. Thus, C is 1ND-detecting. Finally, all the codewords constructed are distinct, hence

$$\|C\| = q^k \|\mathrm{RLL}(n-5k)\| = \|\operatorname{Irr}^{(\mathrm{k})}(n-4k)\| \geqslant \frac{1}{2 \cdot q^{4k}} \cdot M,$$

where the last inequality follows from the fact that $\|\operatorname{Irr}^{(\mathrm{k})}(n-4k)\| \geqslant \|\operatorname{Irr}^{(\mathrm{k})}(n)\|/q^{4k}$ and then from (2.9). □

2.4 Unrestricted error-detecting codes

Substitution mutations might occur not only in k-duplication copies, but also independently in other positions. In what follows, we consider a single substitution error occurring in addition to however

many k-duplications, at any stage during the sequence of k-duplication events, but not necessarily in a duplicated substring. We refer to codes correcting many k-duplication errors and detecting a single independent substitution error as *1S-detecting* codes.

Definition 22. *A code $C \subseteq \Sigma^*$ is a 1S-detecting code if there exists a decoding function $\mathcal{D} : \Sigma^* \to C \cup \{\text{error}\}$ such that if $\boldsymbol{c} \in C$ was transmitted and $\boldsymbol{y} \in \Sigma^*$ was received then $\mathcal{D}(\boldsymbol{y}) = \boldsymbol{c}$ if only k-duplication errors occurred, and $\mathcal{D}(\boldsymbol{y}) \in \{\boldsymbol{c}, \text{error}\}$ if in addition to the k-duplications, exactly one unrestricted substitution occurred.*

Lemma 23. *A code $C \in \Sigma^n$ is a 1S-detecting code if and only if for any two distinct codewords $\boldsymbol{c}_1, \boldsymbol{c}_2 \in C$, we have*

$$\operatorname{drt}(\boldsymbol{c}_1) \neq \operatorname{drt}(\boldsymbol{c}_2) \quad \textit{and} \quad \operatorname{drt}(\boldsymbol{c}_2) \notin \operatorname{drt}(D_k^{*,1}(\boldsymbol{c}_1)). \tag{2.13}$$

Proof. In the one direction, we define for any $\boldsymbol{y} \in \Sigma^*$, $\mathcal{D}(\boldsymbol{y}) = \boldsymbol{c}$ if $\operatorname{drt}(\boldsymbol{c}) = \operatorname{drt}(\boldsymbol{y})$, and $\mathcal{D}(\boldsymbol{y}) = \text{error}$ otherwise. Clearly if (2.13) holds then $\mathcal{D}$ is a decoding function proving that C is a 1S-detecting code.

In the other direction, if (2.5) does not hold we have two (not mutually exclusive) cases. If there exist $\boldsymbol{c}_1, \boldsymbol{c}_2 \in C$ such that $\operatorname{drt}(\boldsymbol{c}_1) = \operatorname{drt}(\boldsymbol{c}_2)$ then by Theorem 2 there exists $\boldsymbol{y} \in D_k^{*,0}(\boldsymbol{c}_1) \cap D_k^{*,0}(\boldsymbol{c}_2)$ and no decoding function can always correctly decode $\boldsymbol{y}$. Similarly, if $\operatorname{drt}(\boldsymbol{c}_2) \in \operatorname{drt}(D_k^{*,1}(\boldsymbol{c}_1))$ then there exists $\boldsymbol{y} \in D_k^{*,1}(\boldsymbol{c}_1)$ such that $\operatorname{drt}(\boldsymbol{y}) = \operatorname{drt}(\boldsymbol{c}_2)$ and no decoding function $\mathcal{D}$ can always decode $\boldsymbol{y}$ correctly. □

We shall adopt the same general strategy as the previous section. Namely, we will construct a code based on irreducible words of length n. Descendants whose duplication root is not of length n will be easily detected. Our challenge is therefore to detect errors that do not change the length of the root caused by, what we refer to as, ambiguous substitutions.

Definition 24. *An unrestricted substitution error that changes the root but not the length of the root is called an* ambiguous unrestricted substitution.

As in the previous section, when the duplication length is $k = 1$ there are no ambiguous unrestricted substitutions. In that case $\operatorname{Irr}^{(\mathrm{k})}(n)$ can easily serve as a 1S-detecting code. Thus, we shall focus on the case of $k \geqslant 2$.

Lemma 25. *Let $n \geqslant 2k \geqslant 4$. For any string $x \in \Sigma^n$, let $\boldsymbol{x}' \in \operatorname{drt}(D_k^{*,\leqslant 1}(\boldsymbol{x})) \cap \Sigma^n$ be a string obtained from x via a single ambiguous unrestricted substitution. If*

$$d(\phi(\operatorname{drt}(\boldsymbol{x})), \phi(\operatorname{drt}(\boldsymbol{x}'))) \geqslant 3,$$

then $\bar{\phi}(\operatorname{drt}(\boldsymbol{x}))$ and $\bar{\phi}(\operatorname{drt}(\boldsymbol{x}'))$ differ by a single k-switch error.

Proof. Let $\boldsymbol{x}' \in D_k^{*,\leqslant 1}(\boldsymbol{x})$, where

$$\left|\operatorname{drt}(\boldsymbol{x}')\right| = |\operatorname{drt}(\boldsymbol{x})|, \qquad \text{but} \qquad \operatorname{drt}(\boldsymbol{x}') \neq \operatorname{drt}(\boldsymbol{x}),$$

namely, an ambiguous unrestricted substitution occurred. Let us denote

$$\boldsymbol{y} \triangleq \hat{\phi}(\boldsymbol{x}), \qquad \boldsymbol{z} \triangleq \bar{\phi}(\boldsymbol{x}),$$
$$\boldsymbol{y}' \triangleq \hat{\phi}(\boldsymbol{x}'), \qquad \boldsymbol{z}' \triangleq \bar{\phi}(\boldsymbol{x}').$$

Since k-duplications do not change the root, we assume without loss of generality that no k-duplications occur and only a single substitution occurs. Thus, we can write

$$\boldsymbol{x}' = \boldsymbol{x} + a \cdot \boldsymbol{e}_i, \qquad \boldsymbol{y}\boldsymbol{z} = \boldsymbol{y}'\boldsymbol{z}' + a \cdot \boldsymbol{\epsilon}_i,$$

where i denotes the location of the substitution, and $a \in \Sigma \setminus \{0\}$. Depending on i, a single substitution may result in one or two changed positions in the transform doamin of ϕ. The proof of the claim comprises of many cases, and we start with some simple ones.

In the first simple case, the substitution occurs in the first k positions, namely, $1 \leqslant i \leqslant k$. Since $\phi(\mathrm{drt}(\boldsymbol{x}')) = \boldsymbol{y}'\mu(\boldsymbol{z}')$, and $\boldsymbol{y} \neq \boldsymbol{y}'$, if we have $|\mathrm{drt}(x')| = |\mathrm{drt}(x)|$ then

$$d(\phi(\mathrm{drt}(\boldsymbol{x})), \phi(\mathrm{drt}(\boldsymbol{x}'))) \leqslant 2,$$

by virtue of positions i and $i + k$.

In a similar fashion, if the substitution occurs in the last k positions, namely, $|\boldsymbol{x}|-k+1 \leqslant i \leqslant |\boldsymbol{x}|$, only a single position is changed in the transform ϕ. Since $\phi(\mathrm{drt}(\boldsymbol{x}')) = \boldsymbol{y}'\mu(\boldsymbol{z}')$, and $\boldsymbol{z} \neq \boldsymbol{z}'$, if we have $|\mathrm{drt}(\boldsymbol{x}')| = |\mathrm{drt}(\boldsymbol{x})|$ then

$$d(\phi(\mathrm{drt}(\boldsymbol{x})), \phi(\mathrm{drt}(\boldsymbol{x}'))) \leqslant 1,$$

by virtue of positions i.

We are now left with the last interesting case, in which the substitution changes two positions, i and $i+k$, both in the $\boldsymbol{z}$ part of the ϕ-transform. We therefore disregard the part $\boldsymbol{y} = \boldsymbol{y}'$. We may now write

$$\begin{aligned} \boldsymbol{z} &= \boldsymbol{u} \quad a_1 \quad \boldsymbol{v} \quad a_2 \quad \boldsymbol{w} \\ z' &= \boldsymbol{u}\ (a_1 + a)\ \boldsymbol{v}\ (a_2 - a)\ \boldsymbol{w} \end{aligned}$$

where $\boldsymbol{u}, \boldsymbol{w} \in \Sigma^*$, $\boldsymbol{v} \in \Sigma^{k-1}$, $a, a_1, a_2 \in \Sigma$, and $a \neq 0$. We distinguish between two major cases, depending on whether $\boldsymbol{v} = 0^{k-1}$.

Case I: In the first major case we have $\boldsymbol{v} = 0^{k-1}$. Let us write

$$\boldsymbol{u} = \boldsymbol{u}'0^{m_1}, \qquad \boldsymbol{w} = 0^{m_4}\boldsymbol{w}',$$

where all the indicated runs of zeros are maximal. Thus,

$$\begin{aligned} \boldsymbol{z} &= \boldsymbol{u}'\ 0^{m_1} \quad a_1 \quad 0^{k-1} \quad \boldsymbol{a}_2 \quad 0^{m_4}\ \boldsymbol{w}' \\ \boldsymbol{z}' &= \boldsymbol{u}'\ 0^{m_1}\ (a_1 + a)\ 0^{k-1}\ (a_2 - a)\ 0^{m_4}\ \boldsymbol{w}'. \end{aligned}$$

The length of the substring between $\boldsymbol{u}'$ and $\boldsymbol{w}'$ is $m_1 + m_4 + k + 1$ and we note that

$$\left\lfloor \frac{m_1 + m_4 + k + 1}{k} \right\rfloor = \left\lfloor \frac{m_1}{k} \right\rfloor + \left\lfloor \frac{m_4}{k} \right\rfloor + s,$$

where $s \in \{1, 2\}$. We distinguish between the following cases:

1. If $a_1 \neq 0$ and $a_2 \neq 0$:

   1.1 If $a_1 + a \neq 0$ and $a_2 - a \neq 0$

   $$d(\phi(\mathrm{drt}(\boldsymbol{x})), \phi(\mathrm{drt}(\boldsymbol{x}'))) \leqslant 2.$$

   1.2 If exactly one of $a_1 + a$ and $a_2 - a$ is zero, the length of $\mu(\boldsymbol{z}')$ decreases by k.

   1.3 If $a_1 + a = a_2 - a = 0$, the length of $\mu(z')$ decreases by sk.

2. If $a_1 \neq 0$ and $a_2 = 0$:

   2.1 If $a_1 + a \neq 0$, since $a_2 - a \neq 0$ the length of $\mu(\boldsymbol{z}')$ increases by k.

   2.2 If $a_1 + a = 0$, since $a_2 - a \neq 0$

   $$d(\phi(\mathrm{drt}(\boldsymbol{x})), \phi(\mathrm{drt}(\boldsymbol{x}'))) \leqslant 1.$$

3. If $a_1 = 0$ and $a_2 \neq 0$:

   3.1 If $a_2 - a \neq 0$, since $a_1 + a \neq 0$ the length of $\mu(\boldsymbol{z}')$ increases by k.

   3.2 If $a_2 - a = 0$, since $a_1 + a \neq 0$

   $$d(\phi(\mathrm{drt}(\boldsymbol{x})), \phi(\mathrm{drt}(\boldsymbol{x}'))) \leqslant 1.$$

4. If $a_1 = a_2 = 0$, the length of $\mu(\boldsymbol{z}')$ increases by sk.

Case II: In the second major case, assume $v \neq 0^{k-1}$. Let us write

$$\boldsymbol{u} = \boldsymbol{u}'0^{m_1}, \qquad \boldsymbol{v} = 0^{m_2}\boldsymbol{v}'0^{m_3}, \qquad \boldsymbol{w} = 0^{m_4}\boldsymbol{w}',$$

where all the indicated runs of zeros are maximal. Let $c \in \Sigma \setminus \{0\}$ be some nonzero letter in $\boldsymbol{v}'$, important to us only for the purpose of being able to refer to the part of the string left of c and the part of the string to the right of c.

1. Examining the part of the string to the left of c:

   1 If $a_1 \neq 0$:

      i. If $a_1 = -a$:

         A. If $\left\lfloor \frac{m_1+m_2+1}{k} \right\rfloor > \left\lfloor \frac{m_1}{k} \right\rfloor$, the length before c decreases by k and the substring $0^{j-1}(-a)0^{k-j}$ is deleted.

B. If $\left\lfloor \frac{m_1+m_2+1}{k} \right\rfloor = \left\lfloor \frac{m_1}{k} \right\rfloor$, the length before c stays the same and the substitution $a_1 \to 0$ occurs.

ii. If $a_1 \neq -a$, the length before c stays the same and the substitution $a_1 \to (a_1 + a)$ occurs.

2 If $a_1 = 0$, then $a_1 \neq -a$, and:

i. If $\left\lfloor \frac{m_1+m_2+1}{k} \right\rfloor > \left\lfloor \frac{m_1}{k} \right\rfloor$, the length before c increases by k and $0^{j-1}a0^{k-j}$ is inserted.

ii. If $\left\lfloor \frac{m_1+m_2+1}{k} \right\rfloor = \left\lfloor \frac{m_1}{k} \right\rfloor$, the length before c stays the same and the substitution $0 \to a$ occurs.

2. Examining the part of the string to the right of c:

1 If $a_2 \neq 0$:

i. If $a_2 = a$:

A. If $\left\lfloor \frac{m_3+m_4+1}{k} \right\rfloor > \left\lfloor \frac{m_4}{k} \right\rfloor$, the length after c decreases by k and $0^{t-1}a0^{k-t}$ is deleted.

B. If $\left\lfloor \frac{m_3+m_4+1}{k} \right\rfloor = \left\lfloor \frac{m_4}{k} \right\rfloor$, the length after c stays the same and the substitution $a_2 \to 0$ occurs.

ii. If $a_2 \neq a$, the length after c stays the same and the substitution $a_2 \to (a_2 - a)$ occurs.

2 If $a_2 = 0$, then $a_2 \neq a$, and:

i. If $\left\lfloor \frac{m_3+m_4+1}{k} \right\rfloor > \left\lfloor \frac{m_4}{k} \right\rfloor$, the length after c increases by k and $0^{t-1}(-a)0^{k-t}$ is inserted.

ii. If $\left\lfloor \frac{m_3+m_4+1}{k} \right\rfloor = \left\lfloor \frac{m_4}{k} \right\rfloor$, the length after c stays the same and the substitution $0 \to (-a)$ occurs.

Based on the changes of a_1 and a_2, there are two types of ambiguous unrestricted substitutions:

- Define the sets of cases $A \triangleq \{1(1)\text{iB}, 1(1)\text{ii}, 1(2)\text{ii}\}$ and $B \triangleq \{2(1)\text{iB}, 2(1)\text{ii}, 2(2)\text{ii}\}$. Any substitution scenario from $A \times B$ results in only two changed symbols, hence

$$d(\phi(\mathrm{drt}(\boldsymbol{x})), \phi(\mathrm{drt}(\boldsymbol{x}'))) \leqslant 2.$$

- The scenarios (1(1)iA,2(2)i) and (1(2)i,2(1)iA) are more complex because they involve both an inserted a substring and a deleted substring of length k. Since the two cases are similar, we only show the analysis of the first case (1(1)iA,2(2)i). We therefore have

$$\begin{aligned} \boldsymbol{z} &= \boldsymbol{u}'\ 0^{m_1}\ a\ 0^{m_2}\ \boldsymbol{v}'\ 0^{m_3}\ 0\ 0^{m_4}\ \boldsymbol{w}' \\ \boldsymbol{z}' &= \boldsymbol{u}'\ 0^{m_1}\ 0\ 0^{m_2}\ \boldsymbol{v}'\ 0^{m_3}\ a\ 0^{m_4}\ \boldsymbol{w}' \end{aligned}$$

where we recall that $a \neq 0$, $|\boldsymbol{v}'| \leqslant k-1$, and $\boldsymbol{v}'$ starts and ends with a non-zero letter. Looking at $\mu(\boldsymbol{z}')$ compared with $\mu(\boldsymbol{z})$, the part to the left of $\boldsymbol{v}'$ becomes shorter by k letters, whereas the part to the right of it becomes longer by k letters. In particular, we can write

$$\begin{aligned} \mu(\boldsymbol{z}) &= u''\ 0^{|\boldsymbol{v}'|}\ 0^{m_3}a0^{m_2}\ \ \boldsymbol{v}'\ \ w'' \\ \mu(\boldsymbol{z}') &= \boldsymbol{u}''\ \ \boldsymbol{v}'\ \ 0^{m_3}a0^{m_2}\ 0^{|v'|}\ \boldsymbol{w}'' \end{aligned} \tag{2.14}$$

where $m_2 + m_3 + |v'| + 1 = k$.

Having considered all cases, this last case is the only one in which we have an ambiguous unrestricted substitution in which potentially $d(\phi(\mathrm{drt}(\boldsymbol{x})), \phi(\mathrm{drt}(\boldsymbol{x}'))) \geqslant 3$. The swapping described in (2.14) completes the proof of the claim. □

Our strategy, based on Lemma 25, is to build a code as an intersection of two other component codes. If one component code can detect the swapping of two substrings and the other component code has a minimum Hamming distance of 3 or more, then their intersection is a 1S-detecting code.

Construction 26. *Let q be a prime power, and $\Sigma \triangleq \mathbb{F}_q$ be the finite field of q elements. Let $n \geqslant k \geqslant 2$ and let r be the unique positive integer such that $\frac{q^{r-1}-1}{q-1} < n \leqslant \frac{q^r-1}{q-1}$, namely,*

$$r \triangleq \left\lceil \log_q(n(q-1)+1) \right\rceil.$$

Denote by C^H the $[n, n-r, 3]$ shortened Hamming code over $\mathbb{F}_q$, and by $C_0^H, C_1^H, \ldots, C_{q^r-1}^H$ its q^r cosets. Finally, let p and $C_{i,j}$ be defined as in Construction 17. For all $0 \leqslant i, j < p$ and $0 \leqslant \ell < q^r$, we construct

$$C_{i,j,\ell} \triangleq \{\boldsymbol{c} \in C_{i,j} \mid \phi(\boldsymbol{c}) \in C_\ell^H\}.$$

Theorem 27. *With the setting as in Construction 26, the code $C_{i,j,\ell}$ is a 1S-detecting code. In particular, there exist i, j, ℓ such that*

$$\|C_{i,j,\ell}\| \geqslant \frac{\|\mathrm{Irr}^{(\mathrm{k})}(n)\|}{q^r p^2} \geqslant \frac{\|\mathrm{Irr}^{(\mathrm{k})}(n)\|}{q(n(q-1)+1)(k+1)^2}.$$

Proof. By Construction 17 we have that $C_{i,j} \subseteq \mathrm{Irr}^{(\mathrm{k})}(n)$, hence also $C_{i,j,\ell} \subseteq \mathrm{Irr}^{(\mathrm{k})}(n)$, which implies it can correct any number of k-duplications. Thus, following Lemma 23, it only remains to consider two distinct codewords $\boldsymbol{c}_1, \boldsymbol{c}_2 \in C_{i,j,\ell}$ and show that $\mathrm{drt}(\boldsymbol{c}_2) \notin \mathrm{drt}(D_k^{*,1}(\boldsymbol{c}_1))$, namely consider the case in which a single ambiguous unrestricted substitution occurred as part of the k-duplications.

Assume to the contrary this is not the case. By Lemma 25, if $d(\phi(\boldsymbol{c}_1), \phi(\boldsymbol{c}_2)) \geqslant 3$, then $\bar{\phi}(\boldsymbol{c}_1)$ and $\bar{\phi}(\boldsymbol{c}_2)$ differ by a single k-switch error, and this is contradicts the fact that $C_{i,j}$ detects a single k-switch error in the $\bar{\phi}$ part of the root, a fact that has already been used in Theorem 18. If $d(\phi(\boldsymbol{c}_1), \phi(\boldsymbol{c}_2)) \leqslant 2$, then this contradicts the minimum distance implied by using the shortened Hamming code.

Finally, the existence of the code with the lower bounded size is guaranteed using a simple averaging argument since $\{C_{i,j,\ell} \mid 0 \leqslant i, j < p, 0 \leqslant \ell < q^r\}$ forms a partition of $\mathrm{Irr}^{(\mathrm{k})}(n)$. □

2.5 Restricted error-correcting codes

2.5.1 Motivation

The previous subsection studies the problem of error-detecting codes for k-duplications and one substitution without asymptotic rate loss. we also consider the error correction in the *noisy duplication model*, which is motivated by the abundance of inexact copies in tandem repeat stretches in

genomes [62]. In particular, one open question is whether we can correct noisy duplications without rate loss.

In the noisy duplication channel, two types of errors are possible: i) exact k-duplications, which insert an exact copy of a substring in tandem, such as $\mathsf{ACGTC} \to \mathsf{ACGT\underline{CGT}C}$; and ii) noisy duplications, which insert approximate copies, e.g., $\mathsf{ACGTC} \to \mathsf{ACGT\underline{CTT}C}$. In both cases, the length of the duplication refers to the length of the duplicated substring (3 in our preceding examples). In this chapter, we limit our attention to exact and noisy tandem duplications of length k, referred to as k-TDs and k-NDs, respectively. Furthermore, we only consider noisy duplications where the copy and the original substring differ in one position. In other words, each noisy duplication can be viewed as an exact k-duplication followed by a substitution in the inserted copy.

For codes capable of correcting any number of exact k-duplications, the best possible asymptotic rate (i.e., the limit of the rate for code length $n \to \infty$, as defined in (1.2)) was given in [30] as

$$1 - \frac{(q-1)\log_q e}{q^{k+2}} + o(q^{-k}), \tag{2.15}$$

where $o(q^{-k})$ represents terms whose ratio to q^{-k} vanishes as k becomes larger. In this subsection, we show that the proposed codes have the same asymptotic rate as (2.15), and are thus asymptotically optimal.

This subseciton is organized as follows. The notation and preliminaries are given in Section 2.5.2. In Section 2.5.3, we analyze the error patterns that manifest as the result of passing through the noisy duplication channel. Finally, the code construction and the corresponding code size are presented in Section 2.5.4.

2.5.2 Notation and preliminaries

In this subsection, we recall that the set of descendants obtained through many exact k-TDs and at most P noisy duplications, i.e., at most P substitution errors, can be expressed as

$$D_k^{*(\leqslant P)}(\boldsymbol{x}) = \bigcup_{p=0}^{p=P} \bigcup_{t=p}^{\infty} D_k^{t(p)}(\boldsymbol{x}). \tag{2.16}$$

In this chapter, we limit our attention to $P = 1$. Furthermore, Since the first k elements are not affected by exact or noisy duplications and $\hat{\phi}(\boldsymbol{x}) = \hat{\phi}(\boldsymbol{x}') = \hat{\phi}(\boldsymbol{x}'')$, we focus on changes in $\bar{\phi}(\cdot)$. Based on the *k-discrete-derivative* transform, the substitution changes at most two symbols of $\bar{\phi}(\boldsymbol{x}')$ and can be expressed as

$$\bar{\phi}(\boldsymbol{x}'') = \bar{\phi}(\boldsymbol{x}') + a\boldsymbol{\epsilon}_j, \tag{2.17}$$

where $\boldsymbol{\epsilon}_j = \boldsymbol{e}_{j-k} - \boldsymbol{e}_j$ if $(k+1) \leqslant j \leqslant (|\boldsymbol{x}'| - k)$ and $\boldsymbol{\epsilon}_j = \boldsymbol{e}_{j-k}$ if $(|\boldsymbol{x}'| - k + 1) \leqslant j \leqslant |\boldsymbol{x}'|$. We refer to $\boldsymbol{x}''$ as a *k-ND descendant of* $\boldsymbol{x}$.

According to [30], given a word $\boldsymbol{x} \in \Sigma_q^*$, after many (even infinite) k-TD errors, the string

$(\hat{\phi}(\boldsymbol{x}), \mu(\bar{\phi}(\boldsymbol{x})))$ stays the same. Based on the definition of *duplication root* $\mathrm{drt}(\boldsymbol{x})$, we then have

$$\phi(\mathrm{drt}(\boldsymbol{x})) = (\hat{\phi}(\boldsymbol{x}), \mu(\bar{\phi}(\boldsymbol{x}))). \tag{2.18}$$

Definition 28. *A word $\boldsymbol{x}$ is called* irreducible *if $\mathrm{drt}(\boldsymbol{x}) = \boldsymbol{x}$. The set of all irreducible words of length n is denoted* $\mathrm{Irr}(n)$*, i.e.,* $\mathrm{Irr}^{(\mathrm{k})}(n) = \{\boldsymbol{x} \in \Sigma_q^n | \, \mathrm{drt}(\boldsymbol{x}) = \boldsymbol{x}\}$.

We note that an irreducible word $\boldsymbol{x} \in \Sigma_q^n$ satisfies $\bar{\phi}(\boldsymbol{x}) \in \mathrm{RLL}(n-k)$.

For a word $\boldsymbol{z} \in \Sigma_q^*$, we define its *indicator* $\Gamma(\boldsymbol{z}) : \Sigma_q^* \to \Sigma_2^*$ as $\Gamma(\boldsymbol{z}) = \Gamma_1(\boldsymbol{z}) \cdots \Gamma_{|\boldsymbol{z}|}(\boldsymbol{z})$, where

$$\Gamma_i(\boldsymbol{z}) = \begin{cases} 1, & \text{if } z_i \neq 0, \\ 0, & \text{otherwise.} \end{cases} \quad i = 1, \ldots, |\boldsymbol{z}|. \tag{2.19}$$

Based on (2.17), the substitution in a noisy duplication alters two symbols in $\bar{\phi}(\boldsymbol{x}')$ at distance k. For the purpose of error correction, it will be helpful to rearrange the symbols into k strings such that the two symbols affected by the substitution appear next to each other in one of the strings. More precisely, for $j \in [k]$, we define a *splitting* operation that extracts entries whose position is equal to j modulo k. That is, for $\boldsymbol{u} \in \Sigma_q^n$ and $j \in [k]$, define $\mathrm{Sp}_k(\boldsymbol{u}, j) = \boldsymbol{u}_j = (u_{j1}, u_{j2}, \ldots, u_{j,\lfloor \frac{n-j}{k} \rfloor + 1})$ such that

$$u_{ji} = u_{j+(i-1)k}, \quad 1 \leqslant i \leqslant \left\lfloor \frac{n-j}{k} \right\rfloor + 1. \tag{2.20}$$

For $\boldsymbol{u} \in \Sigma_q^n$, we then define the *interleaving* operation $\mathrm{IL} : \Sigma_q^n \to \Sigma_q^n$ as the concatenation of $\mathrm{Sp}_k(\boldsymbol{u}, j), j \in [k]$,

$$\mathrm{IL}(\boldsymbol{u}) = \mathrm{Sp}_k(\boldsymbol{u}, 1) \cdots \mathrm{Sp}_k(\boldsymbol{u}, k).$$

Example 29. *Given an alphabet $\Sigma_3 = \{0, 1, 2\}$, $k = 3$, and $\boldsymbol{u}' = \bar{\phi}(\boldsymbol{x}') = 221200012$, where symbols at the same position modulo k have the same color, after splitting $\boldsymbol{u}'$, we obtain*

$$\begin{aligned} \boldsymbol{u}'_1 &= \mathrm{Sp}_3(\boldsymbol{u}', 1) = 220, \\ \boldsymbol{u}'_2 &= \mathrm{Sp}_3(\boldsymbol{u}', 2) = 201, \\ \boldsymbol{u}'_3 &= \mathrm{Sp}_3(\boldsymbol{u}', 3) = 102, \\ \mathrm{IL}(\boldsymbol{u}') &= \boldsymbol{u}'_1 \boldsymbol{u}'_2 \boldsymbol{u}'_3 = 220201102. \end{aligned}$$

Based on (2.17), after one substitution error, we may obtain $\boldsymbol{u}'' = \bar{\phi}(\boldsymbol{x}'') = 221201011$, where symbols affected by the substitution error are underlined. We then have

$$\begin{aligned} \boldsymbol{u}''_1 &= \mathrm{Sp}_3(\boldsymbol{u}'', 2) = 201, \\ \boldsymbol{u}''_2 &= \mathrm{Sp}_3(\boldsymbol{u}'', 1) = 220, \\ \boldsymbol{u}''_3 &= \mathrm{Sp}_3(\boldsymbol{u}'', 3) = 111, \\ \mathrm{IL}(\boldsymbol{u}'') &= \boldsymbol{u}''_1 \boldsymbol{u}''_2 \boldsymbol{u}''_3 = 220201111. \end{aligned}$$

We observe that the error is restricted to $\boldsymbol{u}''_3$ and that the two symbols changed by the substitution

error are adjacent in $\mathrm{IL}(\boldsymbol{u}'')$*, while they are not so in* $\boldsymbol{u}''$*.*

Given a word $\boldsymbol{z} \in \Sigma_q^n$, we define the *cumulative-sum* operation $\mathrm{CS} : \Sigma_q^n \to \Sigma_q^n$, as $\boldsymbol{r} = \mathrm{CS}(\boldsymbol{z})$, where

$$r_i = \sum_{t=1}^{i} z_t \bmod q, \quad i = 1, \ldots, n. \tag{2.21}$$

We further define the *odd subsequence* $\mathrm{Od}(\boldsymbol{z})$ and the *even subsequence* $\mathrm{Ev}(\boldsymbol{z})$ of a word $\boldsymbol{z} \in \Sigma_q^*$ as two sequences containing symbols in the odd and even positions, respectively. More precisely, $\mathrm{Od}(\boldsymbol{z}) = \mathrm{Sp}_2(\boldsymbol{z}, 1)$ and $\mathrm{Ev}(\boldsymbol{z}) = \mathrm{Sp}_2(\boldsymbol{z}, 2)$.

Our results will rely on codes that can correct a single insertion or deletion. We thus recall the Varshamov-Tenengolts codes [77], [91], which are binary codes capable of correcting a single insertion or deletion (indel).

Construction 30 ([77])**.** *Given integers* $m \geqslant 1$ *and* $0 \leqslant \alpha \leqslant (m-1)$*, the binary Varshamov-Tenengolts (VT) code* $C_{VT}(\alpha, m)$ *is given as*

$$C_{VT}(\alpha, m) = \{\boldsymbol{z} \in \Sigma_2^{\leqslant m-1} | \sum_{i=1}^{|\boldsymbol{z}|} i z_i = \alpha \bmod m\}.$$

We note that there is a minor difference between this construction and the original VT code. Namely, we allow strings of length at most $m-1$ rather than exactly $m-1$. If the length of the stored word is known, it follows from the proof of the VT code that the code in the construction above can correct a single indel.

Compared to the binary indel-correcting code, correcting indels in non-binary sequences is more challenging. We will use Tenengolts' q-ary single-indel-correcting code [91], which relies on the mapping $\zeta : \Sigma_q^* \to \Sigma_2^*$, where the i-th position of $\zeta(\boldsymbol{z})$ is

$$\zeta_i(\boldsymbol{z}) = \begin{cases} 1, & \text{if } z_i \geqslant z_{i-1}, \\ 0, & \text{if } z_i < z_{i-1}. \end{cases} \qquad i = 2, 3, \ldots, |\boldsymbol{z}|.$$

with $\zeta_1(\boldsymbol{z}) = 1$.

Construction 31 ([91])**.** *For integers* $m \geqslant 1$*,* $0 \leqslant \alpha \leqslant (q-1)$ *and* $0 \leqslant \beta \leqslant (m-1)$*, Tenengolts' q-ary single indel-correcting code* $C_{Tq}(\alpha, \beta, m)$ *over* $\Sigma_q^{\leqslant m}$ *is given as*

$$C_{Tq}(\alpha,\beta, m) = \Bigg\{ \boldsymbol{z} \in \Sigma_q^{\leqslant m} \Bigg| \sum_{j=1}^{|\boldsymbol{z}|} z_j = \alpha \bmod q, \\ \sum_{i=1}^{|\boldsymbol{z}|} (i-1)\zeta_i(\boldsymbol{z}) = \beta \bmod m \Bigg\}.$$

Again, we allow codewords of length at most m, rather than exactly m as was the case in Tenengolts' original construction. If the length of the stored codeword is known, it follows from the

proofs of the VT code that we can recover the binary sequence $\zeta_i(\boldsymbol{z})$ with $|\boldsymbol{z}| \leqslant m$. Then the code in the construction above can correct a single indel in $\boldsymbol{z}$.

In contrast to Lemma 25 and Construction 26, we shall see in the following example that an intersection of a single substitution correcting code with a k-duplication correcting code is not, in general, a code that can correct k-duplications and one substitution.

Example 32. *Set $\Sigma = \mathbb{Z}_2$ and $k = 3$, and observe the following two sequences of k-duplication and substitution, as seen in the ϕ-transform domain:*

$$\begin{aligned}\boldsymbol{u} &\triangleq 111010111 \to 111010111\underline{000} \to 111\underline{0}0101\underline{0}1000\\ \boldsymbol{v} &\triangleq 111101010 \to 111\underline{000}101010 \to 111000101\underline{0}0\end{aligned}$$

It is clear that if $C \subseteq \Sigma^{\geqslant k}$ is a code correcting even a single k-duplication and a single substitution, even given the order in which they occur, then $\phi^{-1}(\boldsymbol{u}) = 111101010$ and $\phi^{-1}(\boldsymbol{v}) = 111010000$ cannot both belong to C. Observing that $\boldsymbol{u}, \boldsymbol{v} \in \mathrm{RLL}(9)$ and the Hamming distance is $d_H(\phi^{-1}(\boldsymbol{u}), \phi^{-1}(\boldsymbol{v})) = 4$, however, we find that $C \triangleq \{\phi^{-1}(\boldsymbol{u}), \phi^{-1}(\boldsymbol{v})\}$ can correct any number of k-duplications, or correct a single substitution. Simple intersections, hence, do not suffice for a code correcting a combination of such errors.

2.5.3 Noisy duplication channels

To enable designing error-correcting codes, in this section, we study the relation between the input and output sequences in *noisy duplication channels*. As before, we consider channels with many (possibly infinite) exact k-duplications and at most one noisy duplication in which one of the copied symbols is altered.

If a code $C \in \Sigma_q^n$ corrects many k-TD and one k-ND errors, then for any two distinct codewords $\boldsymbol{c}_1, \boldsymbol{c}_2 \in C$, we have

$$D_k^{*(\leqslant 1)}(\boldsymbol{c}_1) \cap D_k^{*(\leqslant 1)}(\boldsymbol{c}_2) = \varnothing,$$

where $D_k^{*(\leqslant 1)}(\cdot)$ is defined in (2.16). This can be shown to be equivalent to

$$\begin{aligned}\mathrm{drt}(\boldsymbol{c}_2) &\neq \mathrm{drt}(\boldsymbol{c}_1),\\ \mathrm{drt}(D_k^{*(\leqslant 1)}(\boldsymbol{c}_1)) \cap \mathrm{drt}(D_k^{*(\leqslant 1)}(\boldsymbol{c}_2)) &= \varnothing.\end{aligned} \tag{2.22}$$

Since k-TDs do not alter the root of the sequence, $\mathrm{drt}(\boldsymbol{c}_2) \neq \mathrm{drt}(\boldsymbol{c}_1)$ ensures that k-TD errors can be corrected. Noisy tandem duplications however alter the roots. In fact, they may produce sequences with roots whose lengths are different from the roots of the stored sequences. Since the codewords have distinct roots, it suffices to recover the root of the retrieved word to correct any errors. We will restrict our constructions to codes whose codewords are irreducible, and thus are their own roots. While this is not necessary, it will simplify the code construction, as we will show, and does not incur a large penalty in terms of the size of the code.

For noisy duplication channels, given a codeword $\boldsymbol{x} \in \Sigma_q^n$, the generation of descendants $\boldsymbol{x}'' \in D_k^{*(\leqslant 1)}(\boldsymbol{x})$ includes three different cases: only k-TDs; k-TDs followed by one k-ND; and k-TDs,

followed by a k-ND, followed by more k-TDs. Since the root is not affected by the k-TDs, to study $\mathrm{drt}(D_k^{*(\leqslant 1)}(\boldsymbol{x}))$, we only need to consider the second case, i.e., we focus on descendants $\boldsymbol{x}''$ immediately after the noisy duplication.

Given an irreducible string $\boldsymbol{x} \in \Sigma_q^n$ with $n > 2k$, our goal is to characterize $\mathrm{drt}(D_k^{*(\leqslant 1)}(\boldsymbol{x}))$. Based on (2.2), we have

$$\phi(\boldsymbol{x}) = (\hat{\phi}(\boldsymbol{x}), \bar{\phi}(\boldsymbol{x})) = (\boldsymbol{y}, \boldsymbol{z}),$$

where $\boldsymbol{y} = \hat{\phi}(\boldsymbol{x}) \in \Sigma_q^k$ and $\boldsymbol{z} = \bar{\phi}(\boldsymbol{x}) \in \Sigma_q^{n-k}$. Since $\boldsymbol{x}$ is an irreducible string, the string $\boldsymbol{z}$ contains no runs of 0^k, i.e. $\boldsymbol{z} = \mu(\boldsymbol{z})$.

After many k-TDs and one k-ND, we have a descendant $\boldsymbol{x}'' \in D_k^{*(\leqslant 1)}(\boldsymbol{x})$. Since the substitution only occurs in the copy, the first k symbols always stay the same. Thus $\boldsymbol{x}''$ satisfies

$$\phi(\boldsymbol{x}'') = (\hat{\phi}(\boldsymbol{x}''), \bar{\phi}(\boldsymbol{x}'')) = (\hat{\phi}(\boldsymbol{x}), \bar{\phi}(\boldsymbol{x}'')) = (\boldsymbol{y}, \boldsymbol{z}'').$$

Based on (2.18), it suffices to study the problem in the transform domain, i.e., we want to obtain all possible $(\boldsymbol{y}, \mu(\boldsymbol{z}''))$ derived from $(\boldsymbol{y}, \mu(\boldsymbol{z}))$. Our code constructions in the next section will also rely on certain sequences derived from $\mu(\boldsymbol{z})$. The next theorem characterizes how these sequences can be altered by k-TDs and one k-ND. The theorem relies on the indicator map Γ, defined in (2.19), and on the splitting operation defined in (2.20).

Theorem 33. *Let $\boldsymbol{x} \in \Sigma_q^n$ and let $\boldsymbol{x}'' \in D_k^{*(\leqslant 1)}(\boldsymbol{x})$ be a descendent of $\boldsymbol{x}$ (produced by passing through the noisy duplication channel). Furthermore, let*

$$\boldsymbol{z} = \bar{\phi}(\boldsymbol{x}), \qquad \boldsymbol{\mu} = \mu(\boldsymbol{z}),$$
$$\boldsymbol{\mu}_j = \mathrm{Sp}_k(\boldsymbol{\mu}, j), \qquad \boldsymbol{s}_j = \Gamma(\boldsymbol{\mu}_j),$$

Then we define $\boldsymbol{z}'', \boldsymbol{\mu}'', \boldsymbol{\mu}_j'', \boldsymbol{s}_j''$, similarly, based on $\boldsymbol{x}''$. The differences between sequences defined based on $\boldsymbol{x}$ and $\boldsymbol{x}''$ are given in Table 2.3 and Table 2.4.

Proof. In a noisy duplication channel with many exact k-TDs and at most one k-ND, given a string $\boldsymbol{x} \in \Sigma_q^*$, let $\phi(\boldsymbol{x}) = (\boldsymbol{y}, \boldsymbol{z})$ with $\boldsymbol{y} = \hat{\phi}(\boldsymbol{x}) \in \Sigma_q^k$ and $\boldsymbol{z} = \bar{\phi}(\boldsymbol{x}) \in \Sigma_q^*$. Since the k-TDs do not change the duplication root $\mathrm{drt}(\boldsymbol{x})$, we focus our attention to the substitution that will change the duplication root. After many exact k-TDs, we obtain $\boldsymbol{x}' \in D_k^{\geqslant 1(0)}(\boldsymbol{x})$, a descendant of $\boldsymbol{x}$. After the substitution error, we have $\boldsymbol{x}'' \in D_k^{\geqslant 1(1)}(\boldsymbol{x})$. Since the following k-TD errors do not change the duplication root $\mathrm{drt}(\boldsymbol{x}'')$, we focus on the descendants $\boldsymbol{x}'$ and $\boldsymbol{x}''$.

Let $\phi(\boldsymbol{x}') = (\boldsymbol{y}, \boldsymbol{z}')$ and $\phi(\boldsymbol{x}'') = (\boldsymbol{y}, \boldsymbol{z}'')$. In the transform domain, the string $\boldsymbol{z}'$ can be expressed as

$$\boldsymbol{z}' = \boldsymbol{u} a_1 a_2 \cdots a_i \cdots a_k b_1 b_2 \cdots b_i \cdots b_k \boldsymbol{v}.$$

where $\boldsymbol{u}, \boldsymbol{v} \in \Sigma_q^*$ and $a_i, b_i \in \Sigma_q, i \in [k]$. Let the length of the run of 0s on the left side of a_i be m_1 and on the right side of a_i be m_2 (ending at b_i and excluding a_i, b_i), i.e., the substring $c0^{m_1}a_i0^{m_2}d$ with $a, b \in \Sigma_q^+$. Similarly, we define m_3 and m_4 as the length of the run of 0s on the left side and right side of b_i, starting from a_i and excluding a_i, b_i. Based on (2.17), if the substitution position

Table 2.3: The changes in $\boldsymbol{\mu}_j$ and $\boldsymbol{s}_j$, $j \in [k]$ as a result of exact and noisy duplications, when the position of the substitution in $\boldsymbol{x}''$ satisfies $k < p \leqslant (|\boldsymbol{x}''| - k)$. Here $a, b, c \in \Sigma_q$, $d \in \Sigma_2$, $\bar{a} = -a$, and $a, b \neq 0$. Furthermore, $\Lambda \to \boldsymbol{u}$ and $\boldsymbol{u} \to \Lambda$ represent insertion and deletion of the string $\boldsymbol{u}$, respectively. Rows marked by $(*)$ indicate that this type of error occurs for at most one value of $j \in [k]$. The marking (\$) is related to the error-correction strategy discussed in Section 2.5.4.

$\lvert\boldsymbol{\mu}''\rvert - \lvert\boldsymbol{\mu}\rvert$	$\boldsymbol{\mu} \to \boldsymbol{\mu}''$	$\boldsymbol{\mu}_j \to \boldsymbol{\mu}''_j$	$\boldsymbol{s}_j \to \boldsymbol{s}''_j$
$+2k$	insert $0^{j-1}a0^{k-j}$ and $0^{t-1}(0-a)0^{k-t}$	$\Lambda \to a\bar{a}$ $(*)$ $\Lambda \to 00$ (\$) $c \to 0c0$ (\$)	$\Lambda \to 11$ $\Lambda \to 00$ $d \to 0d0$
$+k$	insert $0^{j-1}a0^{k-j}$ and substitute $b_i \to (b_i - a)$	$c \to a(c-a), c \neq a$ $(*)$ $a \to a0$ $(\Lambda \to 0)$ (\$) $\Lambda \to 0$ (\$)	$0 \to 11, 1 \to 11$ $1 \to 10$ $(\Lambda \to 0)$ $\Lambda \to 0$
	substitute $0 \to a$ and insert $0^{t-1}(0-a)0^{k-t}$	$0 \to a\bar{a}$ $(*)$ $\Lambda \to 0$ (\$)	$0 \to 11$ $\Lambda \to 0$
0	insert $0^{j-1}a0^{k-j}$ and delete $0^{t-1}a0^{k-t}$ with a at the same position	$b0 \to 0b$ (\$) stay same	$10 \to 01$ stay same
	substitute $0 \to a$ and $b_i \to (b_i - a)$ with distance k	$0c \to a(c-a)$ $(*, \$)$ stay same	$00 \to 11, 01 \to 11, 01 \to 10$ stay same
$-k$	substitute $0 \to a$ and delete $0^{t-1}a0^{k-t}$	$0 \to \Lambda$ (\$)	$0 \to \Lambda$

Table 2.4: The changes in $\boldsymbol{\mu}_j$ and $\boldsymbol{s}_j$, $j \in [k]$ as a result of exact and noisy duplication, when the position of the substitution in $\boldsymbol{x}''$ satisfies $(|\boldsymbol{x}''| - k) < p \leqslant |\boldsymbol{x}''|$. The notation is the same as that of Table 2.3.

$\lvert\boldsymbol{\mu}''\rvert - \lvert\boldsymbol{\mu}\rvert$	$\boldsymbol{\mu} \to \boldsymbol{\mu}''$	$\boldsymbol{\mu}_j \to \boldsymbol{\mu}''_j$	$\boldsymbol{s}_j \to \boldsymbol{s}''_j$
$+k$	insert $0^{j-1}a0^{k-j}$	$\Lambda \to a$ $(*)$ $\Lambda \to 0$ (\$)	$\Lambda \to 1$ $\Lambda \to 0$
0	substitute $0 \to a$	$0 \to a$ $(*, \$)$ stay same	$0 \to 1$ stay same

p satisfies $k < p \leqslant (|\boldsymbol{x}'| - k)$, the substitution changes two symbols; if $(|\boldsymbol{x}'| - k) < p \leqslant |\boldsymbol{x}'|$, the substitution changes one symbol.

First, we consider the substitution position satisfying $k < p \leqslant (|\boldsymbol{x}'| - k)$ such that two symbols of $\boldsymbol{z}'$ changes. The 2 symbols in $\boldsymbol{z}'$ have a distance of k. After the substitution, we have

$$\boldsymbol{z}'' = \boldsymbol{u}a_1a_2\cdots(a_i+a)\cdots a_kb_1b_2\cdots(b_i-a)\cdots b_k\boldsymbol{v},$$

where $a \in \Sigma_q^+$. Based on (2.3), since the substitution only occurs in the copy of a k-TD, we have $a_i = 0$ and $m_1 + m_2 + 1 \geqslant k$.

Since the length between a_i and b_i is k, we have two cases for m_2 and m_3:

- If $m_2 + m_3 < k$, then $m_2 < (k-1)$ and $m_3 < (k-1)$, which means that the substring between a_i and b_i must contain at least one non-zero symbol.
- If $m_2 + m_3 \geqslant k$, then $m_2 = m_3 = (k-1)$, which means that the substring between a_i and b_i is 0^{k-1}.

A) Descendants with $m_2 + m_3 < k$: Since the substring between a_i and b_i must contain at least one non-zero symbol, the changes in $\mu(\boldsymbol{z}')$, as well as $\mu(\boldsymbol{z})$, caused by a_i and b_i, can be analyzed independently. If the non-zero symbol is $d \in \Sigma_q^+$, with a_i and b_i on the left and right side respectively, the changes in $\mu(\boldsymbol{z}')$ can be separately studied on the two sides of d. In the following, we use $0^{j-1}a0^{k-j}$ or $0^{t-1}a0^{k-t}$ to denote a substring of length k with $\mathrm{wt}(0^{j-1}a0^{k-j}) = \mathrm{wt}(0^{t-1}a0^{k-t}) = 1$, where $j, t \in [k]$ and $a \in \Sigma_q^+$.

1. The changes on the left side of d is caused by changing a_i. Since $a_i = 0$, then $a = a_i + a \neq 0$.
   - 1.1 If $\left\lfloor \frac{m_1+m_2+1}{k} \right\rfloor > \left\lfloor \frac{m_1}{k} \right\rfloor$, the length before d increases by k and the substring $0^{j-1}a0^{k-j}$ is inserted in $\mu(\boldsymbol{z}')$, before the symbol d.
   - 1.2 If $\left\lfloor \frac{m_1+m_2+1}{k} \right\rfloor = \left\lfloor \frac{m_1}{k} \right\rfloor$, the length before d stays the same and 0 is substituted by a at a_i.
2. The changes on the right side of d is caused by changing b_i.
   - 1 If $b_i \neq 0$,
     - i. if $b_i - a = 0$,
       - A. if $\left\lfloor \frac{m_3+m_4+1}{k} \right\rfloor > \left\lfloor \frac{m_4}{k} \right\rfloor$, the length of $\mu(\boldsymbol{z}')$ after d decreases by k and a substring $0^{t-1}a0^{k-t}$ is deleted from $\mu(\boldsymbol{z}')$.
       - B. if $\left\lfloor \frac{m_3+m_4+1}{k} \right\rfloor = \left\lfloor \frac{m_4}{k} \right\rfloor$, the length after d stays the same and a is substituted by 0 at b_i.
     - ii. if $b_i - a \neq 0$, the length after d stays the same and b_i is substituted by $(b_i - a)$.
   - 2 If $b_i = 0$, then $b_i - a \neq 0$.
     - 1 if $\left\lfloor \frac{m_3+m_4+1}{k} \right\rfloor > \left\lfloor \frac{m_4}{k} \right\rfloor$, the length of $\mu(\boldsymbol{z}')$ after d increases by k and the substring $0^{t-1}(0-a)0^{k-t}$ is inserted in $\mu(\boldsymbol{z}')$.

Table 2.5: The changes in $\mu(\boldsymbol{z})$ with $m_2 + m_3 < k$.

a_i and b_i	$\lvert\boldsymbol{\mu}''\rvert - \lvert\boldsymbol{\mu}\rvert$	$\boldsymbol{\mu} \to \boldsymbol{\mu}''$
1.1 and 2(1)iA	0	insert $0^{j-1}a0^{k-j}$ and delete $0^{t-1}a0^{k-t}$
1.1 and 2(1)iB	$+k$	insert $0^{j-1}a0^{k-j}$ and $a \to 0$
1.1 and 2(1)ii	$+k$	insert $0^{j-1}a0^{k-j}$ and $b_i \to (b_i - a)$
1.1 and 1	$+2k$	insert $0^{j-1}a0^{k-j}$ and $0^{t-1}(q-a)0^{k-t}$
1.1 and 2	$+k$	insert $0^{j-1}a0^{k-j}$ and $0 \to (0 - a)$
1.2 and 2(1)iA	$-k$	$0 \to a$ and delete $0^{t-1}a0^{k-t}$
1.2 and 2(1)iB	0	two substitutions ($0 \to a$ and $a \to 0$)
1.2 and 2(1)ii	0	two substitutions($0 \to a$ and $b_i \to (b_i - a)$)
1.2 and 1	$+k$	$0 \to a$ and insert $0^{t-1}(0-a)0^{k-t}$
1.2 and 2	0	two substitutions($0 \to a$ and $0 \to (0 - a)$)

2 if $\left\lfloor \frac{m_3 + m_4 + 1}{k} \right\rfloor = \left\lfloor \frac{m_4}{k} \right\rfloor$, the length after d stays the same and 0 is substituted by $(0 - a)$ at b_i.

Since $\boldsymbol{\mu} = \mu(\boldsymbol{z})$ and $\mu(\boldsymbol{z}) = \mu(\boldsymbol{z}')$, the changes from $\boldsymbol{\mu} = \mu(\boldsymbol{z}')$ to $\boldsymbol{\mu}'' = \mu(\boldsymbol{z}'')$ are shown in Table 2.5 classified based on a_i and b_i.

B) Descendants with $m_2 + m_3 > k$: Based on the analysis above, when $m_2 + m_3 > k$, the substring between a_i and b_i is 0^{k-1}. Hence $\boldsymbol{z}'$ can be rewritten as

$$\boldsymbol{z}' = \boldsymbol{u}0^{m_1}a_i0^{k-1}b_i0^{m_4}\boldsymbol{v},$$

where $\boldsymbol{u}, \boldsymbol{v} \in \Sigma_q^*$. After one substitution, $\boldsymbol{z}''$ can be expressed as

$$\boldsymbol{z}'' = \boldsymbol{u}\underline{0^{m_1}(a_i + a)0^{k-1}(b_i - a)0^{m_4}}\boldsymbol{v},$$

where $a_i = 0$ and $a \in \Sigma_q^+$. Since the length of $\mu(\boldsymbol{z}')$ is influenced by the *underlined substring* above, we focus on the changes of this segment.

The length of the underlined substring satisfies

$$\left\lfloor \frac{m_1 + m_4 + k + 1}{k} \right\rfloor = \left\lfloor \frac{m_4}{k} \right\rfloor + \left\lfloor \frac{m_1}{k} \right\rfloor + 1,$$

or

$$\left\lfloor \frac{m_1 + m_4 + k + 1}{k} \right\rfloor = \left\lfloor \frac{m_4}{k} \right\rfloor + \left\lfloor \frac{m_1}{k} \right\rfloor + 2.$$

The two cases are discussed below in detail.

If the length of the underlined substring satisfies $\left\lfloor \frac{m_1 + m_4 + k + 1}{k} \right\rfloor = \left\lfloor \frac{m_4}{k} \right\rfloor + \left\lfloor \frac{m_1}{k} \right\rfloor + 1$, then the changes in $\mu(\boldsymbol{z}')$ consist of two cases (based on the change from (a_i, b_i) to $(a_i + a, b_i - a)$):

1. if $(a_i, b_i) = (0, q_i)$ with $q_i \neq 0$, then we again have two cases:

1 if a_i+a, b_i-a are non-zero, the length of $\mu(\boldsymbol{z}')$ increases by k, and the substring $0^{j-1}a0^{k-j}$ is inserted in $\mu(\boldsymbol{z}')$ and b_i is substituted by $b_i - a$.

2 if $(a_i + a, b_i - a) = (q_i, 0)$, we have $\mu(\boldsymbol{z}'') = \mu(\boldsymbol{z}')$.

2. if $(a_i, b_i) = (0, 0)$, then $a_i + a, b_i - a$ are non-zero, the length of $\mu(\boldsymbol{z}')$ increases by k, and the substring $0^{j-1}a0^{k-j}$ is inserted in $\mu(\boldsymbol{z}')$ and 0 is substituted by $(0 - a)$ at b_i.

Similarly, if the length of the underlined substring satisfies $\left\lfloor \frac{m_1 + m_4 + k + 1}{k} \right\rfloor = \left\lfloor \frac{m_4}{k} \right\rfloor + \left\lfloor \frac{m_1}{k} \right\rfloor + 2$, the changes in $\mu(\boldsymbol{z}')$ also contain two cases:

1. if $(a_i, b_i) = (0, q_i)$, then there are two different cases:

   1 if a_i+a, b_i-a are non-zero, the length of $\mu(\boldsymbol{z}')$ increases by k, and the substring $0^{j-1}a0^{k-j}$ is inserted in $\mu(\boldsymbol{z}')$ and b_i is substituted by $b_i - a$.

   2 if $(a_i + a, b_i - a) = (q_i, 0)$, we have $\mu(\boldsymbol{z}'') = \mu(\boldsymbol{z}')$.

2. if $(a_i, b_i) = (0, 0)$, then $a_i + a, b_i - a$ are non-zero, the length of $\mu(\boldsymbol{z}')$ increases by $2k$, and the string $0^{j-1}a0^{k-j}$ and $0^{t-1}(0 - a)0^{k-t}$ are inserted in $\mu(\boldsymbol{z}')$

Since the k-TDs do not change the duplication root, we have $\mathrm{drt}(\boldsymbol{x}) = \mathrm{drt}(\boldsymbol{x}')$ and $\mu(\boldsymbol{z}) = \mu(\boldsymbol{z}')$. Based on the analysis above, the changes in $\mu(\boldsymbol{z})$ caused by one substitution can be divided into four different cases:

- if $|\mu(\boldsymbol{z}'')| = |\mu(\boldsymbol{z})| + 2k$, then $\mu(\boldsymbol{z}'')$ is derived from $\mu(\boldsymbol{z})$ by inserting one $0^{j-1}a0^{k-j}$ and one $0^{t-1}(0 - a)0^{k-t}$. Furthermore, a and $(0 - a)$ have distance k.

- if $|\mu(\boldsymbol{z}'')| = |\mu(\boldsymbol{z})| + k$, then $\mu(\boldsymbol{z}'')$ is derived from $\mu(\boldsymbol{z})$ by either inserting $0^{j-1}a0^{k-j}$ and substituting $b_i \to (b_i - a)$ or inserting $0^{t-1}(0 - a)0^{k-t}$ and substituting $0 \to a$. In both cases, a and $(b_i - a)$ have a distance of k.

- if $|\mu(\boldsymbol{z}'')| = |\mu(\boldsymbol{z})|$, three different cases occur. First, $\mu(\boldsymbol{z}'') = \mu(\boldsymbol{z})$, there are no changes. Second, $\mu(\boldsymbol{z}'')$ is derived from $\mu(\boldsymbol{z})$ by two substitutions ($0 \to a$ and $b_i \to (b_i - a)$ with distance k). Third, the string $0^{j-1}a0^{k-j}$ is inserted and $0^{t-1}a0^{k-t}$ is deleted, where a stays in the same position. In the third case, $\mu(\boldsymbol{z}'')$ is derived from $\mu(\boldsymbol{z})$ by swapping 0^e with a substring (the form of d or $c\Sigma_q^{e-2}d$ with $e \neq 0$ and $c, d \in \Sigma_q^+$) between $a_i = 0$ and $b_i = a$, where the distance of the beginning of the two substrings is k. Furthermore, the integer e satisfies $1 \leqslant e \leqslant (k - 1)$.

- if $|\mu(\boldsymbol{z}'')| = |\mu(\boldsymbol{z})| - k$, $\mu(\boldsymbol{z}'')$ is derived from $\mu(\boldsymbol{z})$ by deleting $0^{t-1}a0^{k-t}$ and substituting $0 \to a$.

In conclusion, the changes from $\boldsymbol{\mu} = \mu(\boldsymbol{z})$ to $\boldsymbol{\mu}'' = \mu(\boldsymbol{z}'')$ caused by one substitution are described in the first and second columns of Table 2.3. We now discuss the changes in $\boldsymbol{\mu}_j$, i.e., the difference between $\boldsymbol{\mu}_j$ and $\boldsymbol{\mu}''_j$ for $j \in [k]$. This is done by considering four cases:

- If $|\mu(\boldsymbol{z}'')| = |\mu(\boldsymbol{z})| + 2k$, $\mu(\boldsymbol{z}'')$ is derived from $\mu(\boldsymbol{z})$ by inserting a $0^{j-1}a0^{k-j}$ and a $0^{t-1}(0-a)0^{k-t}$. For $j \in [k]$, the length of each $\boldsymbol{\mu}_j$ increases by 2. For one value of j, $a(0-a)$ is inserted in $\boldsymbol{\mu}_j$ and two 0s are inserted in the other $(k-1)$ strings with a distance at most 2.

- If $|\mu(\boldsymbol{z}'')| = |\mu(\boldsymbol{z})| + k$, $\mu(\boldsymbol{z}'')$ is derived from $\mu(\boldsymbol{z})$ by inserting $0^{j-1}a0^{k-j}$ or $0^{t-1}(0-a)0^{k-t}$ and substituting $(b_i \to (b_i - a))$ or $(0 \to a)$. For $j \in [k]$, the length of $\boldsymbol{\mu}_j$ increases by 1. For one value of j, the insertion and substitution $b_i \to a(b_i - a)$ occur in $\boldsymbol{\mu}_j$ and 0 is inserted into each of the other $(k-1)$ strings.

- If $|\mu(\boldsymbol{z}'')| = |\mu(\boldsymbol{z})|$, $\mu(\boldsymbol{z}'')$ is derived from $\mu(\boldsymbol{z})$ in three different cases. First, $\mu(\boldsymbol{z}'') = \mu(\boldsymbol{z})$, there are no changes. Second, $\mu(\boldsymbol{z}'')$ is derived from $\mu(\boldsymbol{z})$ by substituting two symbols $(0 \to a, b_i \to (b_i - a))$ with distance k. For one value of j, the substitutions $(0b_i \to a(b_i - a))$ occur in $\boldsymbol{\mu}_j$ and the other $(k-1)$ strings stay the same. Third, $\mu(\boldsymbol{z}'')$ is obtained from $\mu(\boldsymbol{z})$ by inserting $0^{j-1}a0^{k-j}$ and deleting $0^{t-1}(0-a)0^{k-t}$. For $j \in [k]$, at least one $\boldsymbol{\mu}_j$ swaps $(b0) \to (0b)$ with $b \in \Sigma_q^+$ and the other strings stay the same.

- If $|\mu(\boldsymbol{z}'')| = |\mu(\boldsymbol{z}'')| - k$, $\mu(\boldsymbol{z}'')$ is derived from $\mu(\boldsymbol{z})$ by deleting $0^{t-1}a0^{k-t}$ and substituting $0 \to a$. For $\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}$, one 0 is deleted from each of the k strings.

The changes of $\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}$ can be summarized in the third column of Table 2.3. The forth column is obtained by noting that $\boldsymbol{s}_j = \Gamma(\boldsymbol{\mu}_j), j \in [k]$. This completes the proof of Table 2.3.

Second, we consider the case in which the substitution position p satisfies $(|\boldsymbol{x}'| - k) < p \leqslant |\boldsymbol{x}'|$, which means that one symbol in $\boldsymbol{z}$ changes. Since one substitution only changes one symbol in $\boldsymbol{z}'$, we have

$$\boldsymbol{z}'' = \boldsymbol{u}a_1a_2 \cdots (a_i + a) \cdots a_k.$$

where $a \in \Sigma_q^+$. Since the substitution only occurs in a tandem duplication copy, we have $a_i = 0$ and $m_1 + m_2 + 1 \geqslant k$. Note that $a = a_i + a \neq 0$. There are two cases to consider:

1. If $\left\lfloor \dfrac{m_1 + m_2 + 1}{k} \right\rfloor > \left\lfloor \dfrac{m_1}{k} \right\rfloor$, then the length of $\mu(\boldsymbol{z}')$ increases by k and the substring $0^{j-1}a0^{k-j}$ is inserted into $\mu(\boldsymbol{z}')$.

2. If $\left\lfloor \dfrac{m_1 + m_2 + 1}{k} \right\rfloor = \left\lfloor \dfrac{m_1}{k} \right\rfloor$, then the length of $\mu(\boldsymbol{z}')$ stays the same and 0 is substituted by a at a_i.

We can then find the difference between $\boldsymbol{\mu}_j$ and $\boldsymbol{\mu}_j''$, and $\boldsymbol{s}_j$ and $\boldsymbol{s}_j''$, $j \in [k]$, which are listed in Table 2.4. This completes the proof of Theorem 33.

□

To illustrate the theorem, we provide an example (in the transform domain).

Example 34. *Consider* $\Sigma_3 = \{0, 1, 2\}$, $k = 3$, *and* $\boldsymbol{\mu} = \mu(\boldsymbol{z}) = \boldsymbol{z} = 120102002120$. *Suppose that after several k-TDs, the descendant is* $\boldsymbol{z}' = 0^3103^3200^31020020^6103^320$. *Next a k-ND may insert a substring* 0^3 *(marked red below) and alter one or two symbols (underlined). Depending on the positions of the k-duplication and substitution, the following cases are possible:*

- *If* $\boldsymbol{z}'' = 0^3\underline{0}201\underline{1}00200^3102002^06^310^320$, *then* $\boldsymbol{\mu}'' = \mu(\boldsymbol{z}'') = 020110020102002120$ *and* $|\boldsymbol{\mu}''| - |\boldsymbol{\mu}| = 2k$, *as in the* 1*st row of Table 2.3.*

- *If* $\boldsymbol{z}'' = 0^310^3200^31020020^610^3\underline{0}2\underline{0}21$, *then* $\boldsymbol{\mu}'' = \mu(\boldsymbol{z}'') = 120102002102021$ *and* $|\boldsymbol{\mu}''| - |\boldsymbol{\mu}| = k$, *as in the* 2*nd row of Table 2.3.*

- *If* $\boldsymbol{z}'' = 0^310^3200^3001\underline{1}0\underline{1}0020^610^320$, *then* $\boldsymbol{\mu}'' = \mu(\boldsymbol{z}'') = 121101002120$ *and* $|\boldsymbol{\mu}''| = |\boldsymbol{\mu}|$, *as in the* 3*rd row of Table 2.3.*

- *If* $\boldsymbol{z}'' = 0^310^3200^3002\underline{1}0\underline{0}0020^610^320$, *then* $\boldsymbol{\mu}'' = \mu(\boldsymbol{z}'') = 122102120$ *and* $|\boldsymbol{\mu}''| - |\boldsymbol{\mu}| = -k$, *as in the* 4*th row of Table 2.3.*

Since the length of $\boldsymbol{\mu}$ can change by $-k$, 0, k, or $2k$, the noisy duplication may manifest as deletions, insertions, or substitutions in $\boldsymbol{\mu}$. Furthermore, the complex error patterns in $\boldsymbol{\mu}$ are simplified when we consider $\boldsymbol{\mu}_j, j \in [k]$. The errors marked by $(*)$ occur for at most one value of j. These correspond to positions affected by the substitution. (Rows marked by (\$) relate to our error-correction strategy and are discussed in the next section.

We note that for correcting any number of exact k-duplications and t noisy duplications, each containing a single substitution, a description of the channel can be obtained based on Tables 2.3 and 2.4. This is because the tables describe the effect of a sequence of many exact k-duplications and one noisy duplication on the root of the sequence (and its derived subsequences) and because a sequence of errors containing t noisy duplications can be divided into t parts, each consisting of a number of exact k-duplications and a single noisy duplication. In particular, the length of the root may change by $-2k, -k, 0, k, 2k, 3k$, or $4k$ for two noisy duplications. If each noisy duplication contains more than one substitution, however, characterizing the channel becomes more challenging as the number of possible cases grows.

Now that we have determined all changes from $(\boldsymbol{y}, \boldsymbol{\mu})$ to $(\boldsymbol{y}, \boldsymbol{\mu}'')$ resulting from passing through the noisy duplication channel, we consider the code design to correct many exact k-TDs and at most one noisy duplication in the next section.

2.5.4 Error-correcting codes for noisy duplication channels

Recall from Section 2.5.3 that we are interested in constructing a code $C \subseteq \mathrm{Irr}^{(\mathrm{k})}(n) \cap \Sigma_q^n$ that can correct many exact k-TDs and at most one noisy duplication. Based on (2.22), for any code that corrects k-TDs, two distinct codewords must have distinct roots. Thus, for a stored codeword $\boldsymbol{x}$ and the retrieved word $\boldsymbol{x}''$, if we can recover the duplication root $\mathrm{drt}(\boldsymbol{x})$ of $\boldsymbol{x}$ from $\boldsymbol{x}''$, we can recover the codeword $\boldsymbol{x}$. But we have made a further simplifying assumption that $C \subseteq \mathrm{Irr}^{(\mathrm{k})}(n)$ and thus $\boldsymbol{x} = \mathrm{drt}(\boldsymbol{x})$.

As shown in Theorem 33, k-duplication errors manifest in various ways in $\mathrm{drt}(\boldsymbol{x}'')$ and its counterpart in the μ-transform domain $\mu(\bar{\phi}(\boldsymbol{x}''))$. Hence, for error correction, we utilize several sequences derived from $\boldsymbol{x}$, including $\boldsymbol{\mu}_j$ and $\boldsymbol{s}_j$, $j \in [k]$, as defined in Theorem 33. Furthermore, we define $\boldsymbol{r} = \mathrm{CS}(\mathrm{IL}(\boldsymbol{\mu}))$ and $\boldsymbol{r}'' = \mathrm{CS}(\mathrm{IL}(\boldsymbol{\mu}''))$. We note that $\boldsymbol{r}$ (similarly $\boldsymbol{r}''$) can be directly found by rearranging the elements $x_{k+1} \cdots x_n$.

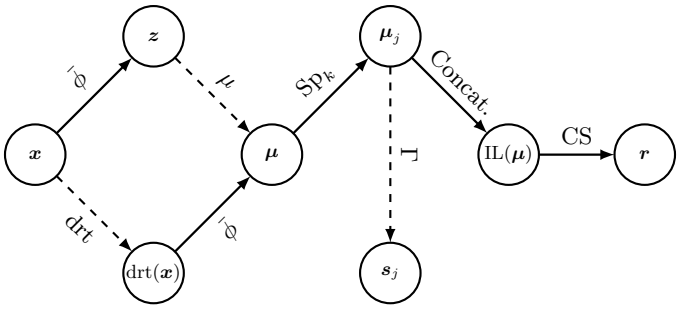


Figure 2.2: The various mapping used in the chapter. "Concat." stands for concatenation. Solid edges indicate invertible mappings, where we have assumed $x_1 \cdots x_k$ is known, since these symbols are not affected by the channel. The mapping μ is generally non-invertible, but in our constructions, since we assume $\boldsymbol{x}$ is irreducible, if we recover $\boldsymbol{\mu} = \mu(\boldsymbol{x})$, we can recover $\boldsymbol{x}$.

The relationship between these mappings is illustrated in Figure 2.2. In the figure, solid edges represent invertible mappings. Since $\boldsymbol{x}$ is irreducible, the stored codeword can be recovered if any of $\boldsymbol{\mu}$, $(\boldsymbol{\mu}_j)_{j\in[k]}$, $\mathrm{IL}(\boldsymbol{\mu})$ or $\boldsymbol{r}$ are recovered (note that $x_1 \cdots x_k$ are not affected by errors). We use these mappings to simplify and correct different error patterns described by Theorem 33 in an efficient manner.

The motivation behind defining $\boldsymbol{\mu}_j$, $j \in [k]$, is to convert insertions and deletions of blocks of length k into simpler errors involving one or two symbols. Some of the errors, marked by (\$) in Tables 2.3 and 2.4, involve 0s, which appear in the same positions in $\boldsymbol{s}_j$ and $\boldsymbol{\mu}_j$. Correcting these errors in $\boldsymbol{s}_j$ is more efficient since it will rely on binary codes rather than q-ary codes. We will first correct these errors in $\boldsymbol{s}_j$ and then correct the corresponding $\boldsymbol{\mu}_j$. Finally, the cumulative-sum mapping CS turns errors marked by $(*)$, e.g., $\Lambda \to a\bar{a}$ into a single q-ary insertion or substitution. Importantly, in each case there is only one such error. So if other errors are corrected, we can concatenate $\boldsymbol{\mu}_j$, $j \in [k]$, and then correct the single occurrence of this error.

We will construct an error-correcting code that will allow us to recover $\boldsymbol{\mu}$ from $\boldsymbol{\mu}''$. As discussed, for certain errors occurring in $\boldsymbol{\mu}_j$, specifically those marked by (\$) in Tables 2.3 and 2.4, we may do so by correcting errors in $\boldsymbol{s}_j$, via Construction 35 below.

The indicator vectors $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k)$ are subject to several error patterns: insertion of 11; insertion of two 0s with distance at most 2; indel of 1 or 0; swaps of two adjacent elements; and substitution of one or two 0s with one or two 1s. The following code can correct a single occurrence of one of these errors, as shown in the next theorem. A slightly modified version of this code is used for the noisy duplication channel.

Construction 35. *Given integers $0 \leqslant a \leqslant 2(n+1)$, $0 \leqslant b \leqslant 4$, and $0 \leqslant c \leqslant 2n$, we construct the code $C_{(a,b,c)}$ as*

$$C_{(a,b,c)} = \{\boldsymbol{u} \in \Sigma_2^n | \boldsymbol{u} \in C_{VT}(a, 2n+3), \tag{2.23}$$

$$\sum_{i=1}^{n} u_i = b \bmod 5, \tag{2.24}$$

$$\sum_{i=1}^{n} i \left(\sum_{j=1}^{j=i} u_j \right) = c \bmod (2n+1)\}, \tag{2.25}$$

where $n = |\boldsymbol{u}|$.

Theorem 36. *The code* $C_{(a,b,c)}$ *can correct a single occurrence of any of the following errors (without a priori knowledge of the type of error):*

- *an insertion, deletion, or substitution,*
- *a substitution of two adjacent bits,*
- *a substitution of one bit by two adjacent bits,*
- *an insertion of two bits of the form* $\Lambda \to 11$, $\Lambda \to 00$, *or* $1 \to 010$.

These error patterns include all those shown in the $\boldsymbol{s}_j$ *column of Tables 2.3 and 2.4.*

Proof. Given a codeword $\boldsymbol{s} \in C_{(a,b,c)}$, let $\boldsymbol{s}''$ be obtained from $\boldsymbol{s}$, either with no error, or via one of the errors listed in Theorem 36.

1. If $|\boldsymbol{s}''| = |\boldsymbol{s}| - 1$, then there has been a single deletion, correctable via the VT code (2.23).

2. If $|\boldsymbol{s}''| = |\boldsymbol{s}|$, then there are the following possibilities: no error, a single substitution, swapping two adjacent different symbols, $00 \to 11$, and $11 \to 00$. Based on (2.24), we have $\sum_{i=1}^{n} s_i'' = (b + b'') \bmod 5$, and b'', along with the syndrome of the VT code, is helpful for distinguishing these cases. If $b'' = 2$, one substitution $00 \to 11$ between $\boldsymbol{s}$ and $\boldsymbol{s}''$ has occurred. We have $\sum_i i s_i'' = a + 2p + 1 \bmod (2n+3)$, where p is the position of the substitution. Hence, we can recover $\boldsymbol{s}$ by one substitution $11 \to 00$ at the position p of $\boldsymbol{s}''$. If $b'' = -2$, a substitution $11 \to 00$ has occurred from $\boldsymbol{s}$ to $\boldsymbol{s}''$. We have $\sum_i i s_i'' = a - 2p - 1 \bmod (2n+3)$. Then we can recover $\boldsymbol{s}$ from $\boldsymbol{s}''$ by flipping two symbols at positions p and $p+1$. If $b'' = 1$, a substitution $0 \to 1$ has occurred. We have $\sum_i i s_i'' = a + p \bmod (2n+3)$. Hence, we can recover $\boldsymbol{s}$ by one substitution $1 \to 0$ at position p of $\boldsymbol{s}''$. If $b'' = -1$, a substitution $1 \to 0$ has occurred. We have $\sum_i i s_i'' = a - p \bmod (2n+3)$. Then $\boldsymbol{s}$ can be recovered by flipping the symbol in the pth position of s''. If $b'' = 0$ and the VT syndrome has changed, an adjacent transposition has occurred in $\boldsymbol{s}$. If the transposition occurs at p, for the constructed string $\{\boldsymbol{s}^{cs} | s_i^{cs} = \sum_{j=1}^{i} s_j, i \in [|\boldsymbol{s}|]\}$, the string $\boldsymbol{s}^{cs}$ and $\boldsymbol{s}^{cs''}$ only differ at position p with $|s_p^{cs} - s_p^{cs''}| = 1$ [19]. Then we have $\sum_i i \left(\sum_{j=1}^{i} s_j'' \right) = c \pm p \bmod (2n+1)$. Thus, we can recover $\boldsymbol{s}$ by swapping the two symbols at positions p and $(p+1)$ of $\boldsymbol{s}''$.

3. If $|\boldsymbol{s}''| = |\boldsymbol{s}| + 1$, based on Theorem 33, $\boldsymbol{s}''$ is derived from $\boldsymbol{s}$ in one of the following ways: inserting a 0, inserting a 1, $0 \to 11$, or $1 \to 00$. Based on (2.23) and (2.24), we have $\sum_i i s_i'' = (a + a'') \bmod (2n+3)$ and $\sum_i s_i'' = (b + b'') \bmod 5$. If $b'' = 0$ and $a'' \leqslant \mathrm{wt}(\boldsymbol{s}'')$, one 0 is inserted in $\boldsymbol{s}$, and we can recover $\boldsymbol{s}$ by deleting it [77]. If $a'' > \mathrm{wt}(\boldsymbol{s}'')$ and $b'' = 1$, one 1 is inserted in $\boldsymbol{s}$. Then we can recover $\boldsymbol{s}$ by deleting a 1 from $\boldsymbol{s}''$ [77]. If $a'' > \mathrm{wt}(\boldsymbol{s}'')$ and

$b'' = 2$, $\boldsymbol{s}''$ is derived from $\boldsymbol{s}$ by a substitution $0 \to 11$. We have $a'' = 2p + 1 + r_1$, where p denotes the position of the original 0 and r_1 denotes the number of 1s on its right. During the recovery process, we denote our guess for the position and the number of 1s on the right side of the position as p' and r_1', respectively. If $r_1' < r_1$, then $2p' + 1 + r_1' > 2p + 1 + r_1$. If $r_1' > r_1$, then $2p' + 1 + r_1' < 2p + 1 + r_1$. Only if $r_1' = r_1$, then $2p' + 1 + r_1' = 2p + 1 + r_1$. If $b'' = -1$, the substitution $1 \to 00$ has occurred. If the substitution is at the position p, then $a'' = -p + r_1$, where r_1 denotes the number of 1s on the right side of the substitution. Similar to correcting the substitution $0 \to 11$, we can obtain the position p and recover $\boldsymbol{s}$ by the substitution $00 \to 1$ at the positions $p, (p+1)$ of $\boldsymbol{s}''$.

4. If $|\boldsymbol{s}''| = |\boldsymbol{s}| + 2$, then $\boldsymbol{s}''$ is derived from $\boldsymbol{s}$ in one of three ways: inserting 11, inserting 00, or inserting two 0s separated by 1 ($1 \to 010$). Based on (2.23) and (2.24), we have $\sum_i i s_i'' = (a + a'') \bmod (2n+3)$ and $\sum_i s_i'' = (b + b'') \bmod 5$. If $b'' = 2$, 11 is inserted in $\boldsymbol{s}$. Let p denote the position in which 11 is inserted. Based on (2.23), we have $a'' = (p+p+1) + 2r_1 = 2(l_0 + l_1 + 1) + 1 + 2r_1 = 2(l_1 + r_1 + 2) + 2l_0 - 1 = 2\,\mathrm{wt}(\boldsymbol{s}'') + 2l_0 - 1$, where l_1 and r_1 denote the number of 1s at the left and right sides of the position p, and l_0 denotes the number of 0s at the left side of the inserting position. Then we can recover $\boldsymbol{s}$ by deleting one 11 from $\boldsymbol{s}''$ after l_0 0s from the beginning. If $b'' = 0$, two 0s are inserted in $\boldsymbol{s}$. If $a'' = 0 \bmod 2$, 00 is inserted in $\boldsymbol{s}$ and $a'' = 2r_1$, where r_1 denotes the number of 1s on the right side of the insertion position. Then we can recover $\boldsymbol{s}$ by deleting 00 from $\boldsymbol{s}''$ before r_1 1s from the end of $\boldsymbol{s}''$. If $a'' = 1 \bmod 2$, two 0s are inserted in $\boldsymbol{s}$ separated by 1 and $a'' = 2r_1 + 1$. Similarly, we can recover $\boldsymbol{s}$ by deleting two 0s before r_1 and $r_1 + 1$ 1s from the end of $\boldsymbol{s}''$.

These error patterns include all those occurring in $\{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k\}$ caused by many exact k-TDs and at most one substitution error in the noisy duplication channel. □

Since $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k)$ are weight indicators of $(\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k)$, the 0s in $(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k)$ and $(\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k)$ coincide. However, if a 1 is deleted from a run of 1s in $\boldsymbol{s}_j$, we will not be able to identify which symbol is deleted from $\boldsymbol{\mu}_j$. This means that after recovering $\boldsymbol{s}_j$ from $\boldsymbol{s}_j''$ we can recover $\boldsymbol{\mu}_j$ only in certain cases, specifically, those marked by (\$) in Table 2.3 and Table 2.4. Interestingly, the errors not corrected by recovering $\boldsymbol{s}_j, j \in [k]$ are marked by $(*)$, indicating that they occur only for a single value of j. Hence, to correct these errors, we apply the code constraints to the concatenation of $\boldsymbol{\mu}_j, j \in [k]$, rather than to each $\boldsymbol{\mu}_j$ separately.

Construction 37. *Define* $C_{nd} \subseteq \Sigma_q^n$ *as*

$$
\begin{aligned}
C_{nd} = \{&\boldsymbol{x} \in \mathrm{Irr}^{(\mathrm{k})}(n) \cap \Sigma_q^n | \boldsymbol{\mu} = \mu(\bar{\phi}(\boldsymbol{x})), \\
&\boldsymbol{\mu}_j = \mathrm{Sp}_k(\boldsymbol{\mu}, j), \boldsymbol{s}_j = \Gamma(\boldsymbol{\mu}_j), \\
&\boldsymbol{s}_j \in C_{VT}(a_j, 2|\boldsymbol{s}_j| + 3), \qquad (2.26) \\
&\sum_{i=1}^{|\boldsymbol{s}_j|} i \left(\sum_{t=1}^{t=i} s_{jt} \right) = c_j \bmod (2|\boldsymbol{s}_j| + 1), \qquad (2.27)
\end{aligned}
$$

$$\sum_{i=1}^{|\boldsymbol{s}_j|} s_{ji} = b \bmod 5, \tag{2.28}$$

$$\mathrm{Od}(\mathrm{IL}(\boldsymbol{\mu})) \in C_{Tq}(\bar{a}_1, \bar{b}_1, \lceil \frac{n-k}{2} \rceil), \tag{2.29}$$

$$\mathrm{Ev}(\mathrm{IL}(\boldsymbol{\mu})) \in C_{Tq}(\bar{a}_2, \bar{b}_2, \lceil \frac{n-k}{2} \rceil), \tag{2.30}$$

$$\mathrm{CS}(\mathrm{IL}(\boldsymbol{\mu})) \in C_{Tq}(\bar{a}_3, \bar{b}_3, n-k), \tag{2.31}$$

$$\mathrm{IL}(\boldsymbol{\mu}) \in C_{Tq}(\bar{a}_4, \bar{b}_4, n-k)\}, \tag{2.32}$$

where $j, a_j, c_j, b, \bar{a}_i, \bar{b}_i$ *are integers satisfying* $j \in [k]$, $0 \leqslant a_j \leqslant 2(|\boldsymbol{s}_j|+1)$, $0 \leqslant c_j \leqslant 2|\boldsymbol{s}_j|$, $0 \leqslant b \leqslant 4$, $0 \leqslant \bar{a}_1, \bar{a}_2, \bar{a}_3, \bar{a}_4 < q$, $0 \leqslant \bar{b}_1, \bar{b}_2 \leqslant \lfloor \frac{n-k}{2} \rfloor$, *and* $0 \leqslant \bar{b}_3, \bar{b}_4 < n-k$.

In Construction 37, the constraints (2.26), (2.27), and (2.28) play the same role as the code in Construction 35, and the constraints (2.29), (2.30), (2.31), and (2.32) can correct the error patterns of $\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}$ not marked by (\$) in Table 2.3 and Table 2.4. The constraint (2.26) corrects one insertion/deletion or two insertions of 0s or 1s in adjacent positions over Σ_2. The constraint (2.27) corrects one transposition of $\{0, 1\}$ in two adjacent positions. The constraint (2.28) is a weight-indicating equation for $\{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k\}$. The constraints (2.29), (2.30), (2.32), and (2.31) can correct one insertion/deletion in $\mathrm{Od}(\mathrm{IL}(\boldsymbol{\mu}))$, $\mathrm{Ev}(\mathrm{IL}(\boldsymbol{\mu}))$, $\mathrm{IL}(\boldsymbol{\mu})$, and $\boldsymbol{r} = \mathrm{CS}(\mathrm{IL}(\boldsymbol{\mu}))$ over Σ_q, respectively.

Theorem 38. *The error-correcting code* C_{nd} *proposed in Construction 37 can correct infinitely many exact* k*-TD and up to one* k*-ND errors. There exists one such code with size*

$$\frac{\|\operatorname{Irr}^{(\mathrm{k})}(n)\|}{5^k q^4 \lceil \frac{n-k}{2} \rceil^2 (4\lceil \frac{n}{k} \rceil^2 - 1)^k (n-k)^2} \leqslant \|C_{nd}\| \leqslant \|\operatorname{Irr}^{(\mathrm{k})}(n)\|. \tag{2.33}$$

Proof. To prove Theorem 38, we have to show that the error-correcting code C_{nd} in Construction 37 can correct all error patterns in $\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}$. Based on Theorem 36, the code $C_{(a,b,c)}$ over Σ_2 can correct all error patterns shown in the $\boldsymbol{\mu}_j$ column of Tables 2.3 and 2.4 in rows marked by (\$). The constraints (2.29), (2.30), (2.31) and (2.32) can correct the other error patterns.

Given a codeword $\boldsymbol{x} \in C_{nd} \subseteq \operatorname{Irr}^{(\mathrm{k})}(n) \cap \Sigma_q^n$, we have $\phi(\mathrm{drt}(\boldsymbol{x})) = (\boldsymbol{y}, \boldsymbol{\mu})$ with $\boldsymbol{y} = \hat{\phi}(\boldsymbol{x}) \in \Sigma_q^k$ and $\boldsymbol{\mu} = \mu(\boldsymbol{z}) = \boldsymbol{z} = \bar{\phi}(\boldsymbol{x}) \in \Sigma_q^{n-k}$. After many exact k-TDs and at most one substitution, we obtain a descendant $\boldsymbol{x}'' \in D_k^{*(\leqslant 1)}(\boldsymbol{x})$ with $\phi(\boldsymbol{x}'') = (\boldsymbol{y}, \boldsymbol{z}'')$ and $\boldsymbol{z}'' = \bar{\phi}(\boldsymbol{x}'')$. In the following, we can recover the codeword $(\boldsymbol{y}, \boldsymbol{\mu})$ by correcting four types of error patterns in $(\boldsymbol{y}, \boldsymbol{\mu}'')$, where $\boldsymbol{\mu}'' = \mu(\boldsymbol{z}'')$. Based on the recovered $(\boldsymbol{y}, \boldsymbol{\mu})$, we can obtain the duplication root $\mathrm{drt}(\boldsymbol{x})$ and thus the codeword $\boldsymbol{x}$. The four cases are below:

- If $|\boldsymbol{\mu}''| = |\boldsymbol{\mu}| - k$, then a 0 is deleted from both $\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}$ and $\{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k\}$. By (2.26), we recover $\{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k\}$ by inserting a 0 in each of them. Based on (2.19), the positions of 0s between $\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}$ and $\{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k\}$ coincide. We can recover $\{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}$ by inserting 0s at the same positions in $\{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_k\}$.

- If $|\boldsymbol{\mu}''| = |\boldsymbol{\mu}|$, $\{\boldsymbol{\mu}_j, j \in [k]\}$ contain two types of errors: transpositions of 0 and b in more than

one $\boldsymbol{\mu}_j$, or the substitution either $0c \to a(c-a)$ or $0 \to a$ in one $\boldsymbol{\mu}_j$. By (2.26), we have

$$\sum_{i=1}^{|\boldsymbol{s}''_j|} i s''_{ji} = (a_j + a''_j) \bmod (2|\boldsymbol{s}_j| + 3), \quad j \in [k].$$

If $\{a''_j, j \in [k]\}$ contain more than one non-zero integer, both $\{\boldsymbol{\mu}_j, j \in [k]\}$ and $\{\boldsymbol{s}_j, j \in [k]\}$ with non-zero $\{a''_j, j \in [k]\}$ contain one adjacent transposition of $(0, b)$ and $(0, 1)$, respectively. By (2.25), the transposition positions $\{p_j, j \in [k]\}$ can be obtained. Since both $\{\boldsymbol{\mu}_j, j \in [k]\}$ and $\{\boldsymbol{s}_j, j \in [k]\}$ contain adjacent transpositions at the same positions, we can recover $\{\boldsymbol{\mu}_j, j \in [k]\}$ by swapping two symbols starting at $\{p_j, j \in [k]\}$. If $\{a''_j, j \in [k]\}$ only contain one non-zero integer, say a''_1, three types of errors may occur based on the weight change of $\{\boldsymbol{s}_j, j \in [k]\}$ by (2.28). Based on the proof of Theorem 36, we can obtain the change position p_1 in $\boldsymbol{\mu}_1$ and $\boldsymbol{s}_1$. If $p_1 < |\boldsymbol{\mu}_1|$, according to Table 2.3, $\boldsymbol{\mu}_1$ contains one substitution $0c \to a(c-a)$, we can recover $\boldsymbol{\mu}_1$ by the substitution $\mu'_{1p_1}\mu'_{1(p_1+1)} \to 0(\mu'_{1p_1} + \mu'_{1(p_1+1)})$. If $p_1 = |\boldsymbol{\mu}_1|$, according to Table 2.4, $\boldsymbol{\mu}_1$ contains one substitution $0 \to a$, we can recover $\boldsymbol{\mu}_1$ by the substitution $\mu'_{1p_1} \to 0$.

- If $|\boldsymbol{\mu}''| = |\boldsymbol{\mu}| + k$, then $(k-1)$ of $\{\boldsymbol{\mu}_j, j \in [k]\}$ contain one insertion $\Lambda \to 0$, and one string, say $\boldsymbol{\mu}_k$, contains either one insertion $\Lambda \to a$ in Table 2.4 or one insertion and one substitution $c \to a(c-a)$ in Table 2.3. By (2.23), the $(k-1)$ strings $\{\boldsymbol{\mu}_j, j \in [k-1]\}$ can be recovered. After that, we generate $\mathrm{IL}'(\boldsymbol{\mu}) = \boldsymbol{\mu}_1 \cdots \boldsymbol{\mu}_{(k-1)}\boldsymbol{\mu}'_k$ by concatenating the k strings. Compared to $\mathrm{IL}(\boldsymbol{\mu})$, $\mathrm{IL}'(\boldsymbol{\mu})$ contains either one insertion $\Lambda \to a$ or one insertion and one substitution $c \to a(c-a)$. Based on (2.29), (2.30), and Construction 31, we obtain the changes $(\Delta\bar{a}_1, \Delta\bar{a}_2)$. If $\Delta\bar{a}_1 + \Delta\bar{a}_2 \neq 0 \bmod q$, then $\mathrm{IL}'(\boldsymbol{\mu})$ contains one insertion $\Lambda \to a$. Then we can recover the insertion $\Lambda \to a$ by (2.32). If $\Delta\bar{a}_1 + \Delta\bar{a}_2 = 0 \bmod q$, $\mathrm{IL}'(\boldsymbol{\mu})$ contains one insertion and one substitution $c \to a(c-a)$. By (2.21) and the fact that $a + (c-a) = c$, we construct $r' = \mathrm{CS}(\mathrm{IL}'(\boldsymbol{\mu}))$ with one insertion. Since (2.31) can correct one insertion in $\mathrm{CS}(\mathrm{IL}'(\boldsymbol{\mu}))$, we can recover $\mathrm{CS}(\mathrm{IL}(\boldsymbol{\mu}))$, $\mathrm{IL}(\boldsymbol{\mu})$, and $\{\boldsymbol{\mu}_j, j \in [k]\}$.

- If $|\boldsymbol{\mu}''| = |\boldsymbol{\mu}| + 2k$, then $(k-1)$ strings of $\{\boldsymbol{\mu}_j, j \in [k]\}$ insert two 0s with distances at most 2, and one string such as $\boldsymbol{\mu}_1$ contains one insertion $a(0-a)$. Similar to the proof of Theorem 36, based on (2.26), we can recover $\{\boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k\}$ by deleting two 0s. After that, we generate the string $\mathrm{IL}'(\boldsymbol{\mu}) = \boldsymbol{\mu}'_1\boldsymbol{\mu}_2 \cdots \boldsymbol{\mu}_k$. Obviously, the string $\mathrm{IL}'(\boldsymbol{\mu})$ contains one insertion $a(0-a)$. When $\mathrm{IL}(\mu(\boldsymbol{z}))$ is divided into two strings $\mathrm{Od}(\mathrm{IL}(\mu(\boldsymbol{z})))$ and $\mathrm{Ev}(\mathrm{IL}(\mu(\boldsymbol{z})))$, one symbol is inserted into each of $\mathrm{Od}(\mathrm{IL}(\mu(\boldsymbol{z})))$ and $\mathrm{Ev}(\mathrm{IL}(\mu(\boldsymbol{z})))$ to generate $\mathrm{Od}(\mathrm{IL}'(\mu(\boldsymbol{z})))$ and $\mathrm{Ev}(\mathrm{IL}'(\mu(\boldsymbol{z})))$. Since both (2.29) and (2.30) can correct an insertion of one symbol in $\mathrm{Od}(\mathrm{IL}(\boldsymbol{\mu}))$ and $\mathrm{Ev}(\mathrm{IL}(\boldsymbol{\mu}))$, respectively, we can recover $\mathrm{IL}(\boldsymbol{\mu})$ and $\{\boldsymbol{\mu}_j, j \in [k]\}$.

Having recovered $\{\boldsymbol{\mu}_j, j \in [k]\}$, we can reconstruct $\boldsymbol{\mu}$, the duplication root $\mathrm{drt}(\boldsymbol{x})$, and the codeword $\boldsymbol{x} \in C_{nd}$. Thus, the error-correcting code C_{nd} can correct all the error patterns caused by many exact k-TD and at most one substitution.

Since $\lceil \frac{n-k}{k} \rceil = \lceil \frac{n}{k} \rceil - 1$, the code size of C_{nd} can be rewritten as

$$\|\operatorname{Irr}^{(\mathrm{k})}(n)\| \geqslant \|C_{nd}\| \geqslant \frac{\|\operatorname{Irr}^{(\mathrm{k})}(n)\|}{5^k q^3 \lceil \frac{n-k}{2} \rceil^2 (4\lceil \frac{n}{k} \rceil^2 - 1)^k (n-k)}.$$

Because the integers $j, a_j, c_j, b, \bar{a}_1, \bar{a}_2, \bar{a}_3, \bar{a}_4, \bar{b}_1, \bar{b}_2, \bar{b}_3, \bar{b}_4$ can be any value in their corresponding ranges, the number of possible codes is $5q^3 \lceil \frac{n-k}{2} \rceil^2 (2\lceil \frac{n-k}{k} \rceil + 3)^k (2\lceil \frac{n-k}{k} \rceil + 1)^k (n-k)$. These codes partition the set $\operatorname{Irr}^{(\mathrm{k})}(n)$, so there is at least one code with size

$$\|C_{nd}\| \geqslant \frac{\|\operatorname{Irr}^{(\mathrm{k})}(n)\|}{5q^4 \lceil \frac{n-k}{2} \rceil^2 (2\lceil \frac{n-k}{k} \rceil + 3)^k (2\lceil \frac{n-k}{k} \rceil + 1)^k (n-k)^2}.$$

Since $\lceil \frac{n-k}{k} \rceil = \lceil \frac{n}{k} \rceil - 1$, the code size of C_{nd} can be rewritten as

$$\|\operatorname{Irr}^{(\mathrm{k})}(n)\| \geqslant \|C_{nd}\| \geqslant \frac{\|\operatorname{Irr}(n)\|}{5^k q^4 \lceil \frac{n-k}{2} \rceil^2 (4\lceil \frac{n}{k} \rceil^2 - 1)^k (n-k)^2}.$$

□

From (2.33), we have

$$\begin{aligned} \frac{1}{n} \log_q \|\operatorname{Irr}^{(\mathrm{k})}(n)\| - \frac{2k+4}{n} \log_q n - \frac{5k+5}{n} \\ \leqslant R_n(C_{nd}) \leqslant \frac{1}{n} \log_q \|\operatorname{Irr}^{(\mathrm{k})}(n)\|. \end{aligned}$$

Furthermore, based on [90, (8)], for $q+k \geqslant 4$, $\frac{M}{2} \leqslant \|\operatorname{Irr}^{(\mathrm{k})}(n)\| \leqslant M$, where $M \triangleq \sum_{i=0}^{\lfloor n/k \rfloor - 1} \|\operatorname{Irr}^{(\mathrm{k})}(n-ik)\|$ is the size of the optimal code of length n that can correct any number of exact k-duplications. Hence,

$$\begin{aligned} \frac{1}{n} \log_q M - \frac{2k+4}{n} \log_q n - \frac{5k+6}{n} \\ \leqslant R_n(C_{nd}) \leqslant \frac{1}{n} \log_q M. \end{aligned}$$

In particular, compared to the optimal code correcting only exact k-duplications, the redundancy is $\lesssim (2k+4) \log_q n$ symbols. Additionally both codes have the same asymptotic rate (given in (2.15) for large k), and in this sense the code proposed here is asymptotically optimal, although it is not clear whether $(2k+4) \log_q n$ is the best possible redundancy.

For the alphabet size $q \in \{3, 4, 5\}$ and the duplication length $k = 3$, Figure 2.3 shows the lower bound of the code rate as the length n of codewords ranges from 100 to 400, based on (2.33), (1.1) and [30].

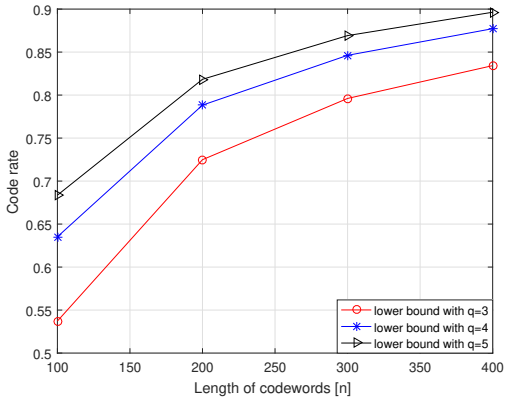


Figure 2.3: The lower bound of the code rate with respect to the length n with the duplication length $k = 3$ and alphabet size $q \in \{3, 4, 5\}$.

2.6 Summary

We have studied the combination of a single substitution error with an unlimited number of tandem-duplication errors, with a fixed duplication-window length. We focused on two noise models, where the substitution error is either restricted to occur in an inserted copy during one of the k-duplication events, or may occur at any position in the string. We have presented bounds as well as constructions of error-detecting and error-correcting codes for both models. In all cases, a rate loss is observed due to the need to recover from an unlimited number of k-duplications. Thus, we are interested in the *extra redundancy cost* due to single-error detection or correction.

In the first case of detecting a single restricted substitution, we show that the additional required cost in redundancy is bounded from above by $\log_q(4(n-k))$ using a GV argument in Theorem 12, where Construction 17 also shows that it is bounded from above by $\log_q(2(k+1)^2)$; depending on the asymptotic regime of k, either may be tighter than the other. In Construction 20 we find a constructive procedure for generating codes for that purpose, which incur a higher redundancy cost of $4k\log_q(2)$; if k is fixed, which is a likely scenario, then that cost is nonetheless constant as well, and improves upon Theorem 12.

Further, in the second case of unrestricted substitution noise, Construction 26 provides error-detecting codes for a single substitution incurring an extra redundancy cost of $O(\log(k^2 n))$ in Theorem 27 by choosing the set of codewords of the intersection of a Hamming code for one substitution and Construction 17 for a k-switch error.

Finally, motivated by reducing the gap of rate loss to correct a noisy duplication, we construct error-correcting codes to correct many exact k-duplications and one noisy duplication, equivalently one *unrestricted substitution*, which suffers from a substitution. Our error-correction strategy is based on roots of sequences and splitting operations that distribute the effect of the k-duplication among k subsequences of the root but bring symbols affected by a substitution together, shown in Theorem 33. We then constructed an error-correcting code Construction 37 that first recovers certain binary substrings with whose help we can determine $k-1$ of the subsequences of the root, leaving

a q-ary error in the last subsequence, which is corrected using Tenengolts' q-ary code. Compared to the optimal code for correcting only exact k-duplications, the proposed construction incurs a redundancy of approximately $(2k+4)\log_q n$ symbols, without an asymptotic rate loss.

Chapter 3

Correcting short tandem duplications and at most p edits

3.1 Introduction

The previous chapter presents a group of error-detecting codes and error-correcting codes that can correct any fixed k-TDs and at most one restricted or unrestricted substitution. Apart from k-TDs, duplications with bounded length k, $\leqslant k$-TDs, are also widely considered in recent works [8], [30], [35]. This chapter focuses on constructing error-correcting codes to correct an arbitrary number of short (tandem) duplications with length bounded by 3, i.e., $\leqslant$3-TDs, and at most p edits, where each edit can be an insertion, deletion, or a substitution. In the rest of this chapter, we call the channels with an arbitrary number of short duplications and at most p substitutions as *DS(p) channels*. Similarly, let *DE(p) channels* denote the channels with an arbitrary number of short duplications and at most p edits. *Furthermore, unless otherwise stated, $\leqslant$3-TDs denote short duplications, and irreducible strings represent $\leqslant$3-irreducible strings over Σ_q.* We start by presenting an simple example of the DS(2) channel and discussing challenges of correct errors.

Example 39. *Given the input* GTCAC*, an arbitrary number of $\leqslant$3-TDs and* 2 *substitutions may generate*

$$\boldsymbol{x} = \mathsf{GTCAC} \to \mathsf{GTC\underline{GTC}AC} \to \mathsf{GTCGACAC} \to$$

$$\mathsf{GTCGA\underline{GA}CAC} \to \mathsf{GTGGAGACAC} \to \mathsf{GTGGA\underline{GG}AGACAC},$$

where the substituted symbols are marked in red, and the short duplication copies are marked with underlines. Note that there are no restrictions on the orders of the occurrences of short duplications and substitutions. Provided that an arbitrary number of $\leqslant$3-TDs are possible, the output may have an unbounded length. Furthermore, a single substitution may be duplicated many times and affect an unbounded segment of the output.

To correct any number of short duplications and at most p edits, this chapter first constructs error-correcting codes capable of correcting any short duplications and at most one edit. After that, a modified error-correcting code is proposed to correct any short duplications and at most p

substitutions. Finally, to reduce the redundancy, we further construct an error-correcting code for the short duplications and at most p edits by applying the syndrome compression technique [88]. When $q \geqslant 4$ and p are constant, the state-of-the-art construction [88] achieves the same asymptotic code rate as the error-correcting code for $\leqslant$3-TDs only [30], with an extra redundancy of roughly $8p \log_q n$ symbols, and has polynomial time complexities in the encoding and decoding processes.

3.2 Notation and preliminaries

We define a *substring edit* in a string $\boldsymbol{x} \in \Sigma_q^*$ as the operation of replacing a substring $\boldsymbol{u}$ with a string $\boldsymbol{v}$, where at least one of $\boldsymbol{u}, \boldsymbol{v}$ is nonempty. The length of the substring edit is $\max\{|\boldsymbol{u}|, |\boldsymbol{v}|\}$. An *$L$-substring edit* is one whose length is at most L. For example, given $\boldsymbol{x} = 0213122013$, a 4-substring edit can generate the sequence $\boldsymbol{y} = 021132013$ by replacing the $\boldsymbol{x}_{[4,6]} = 3120$ by $\boldsymbol{y}_{[4,6]} = 13$. Furthermore, a *burst deletion* in $\boldsymbol{x} \in \Sigma_q^*$ is defined as removing a substring $\boldsymbol{v}$ of $\boldsymbol{x}$, where $|\boldsymbol{v}|$ is the length of the burst deletion. A *$\leqslant L$-burst deletion* has length at most L. For example, given $\boldsymbol{x} = 0213122013$, a $\leqslant$4-burst deletion may generate $\boldsymbol{y} = 0212013$ by removing the substring $\boldsymbol{x}_{[4,7]} = 312$.

Recall that a *(tandem) duplication* (TD) of length k (k-TD) is an operation of generating a copy of a substring and inserting it directly following the substring, where k is the length of the copy. For example, for a string $\boldsymbol{x} = \boldsymbol{uvw}$ with $|\boldsymbol{v}| = k$, a k-TD may generate $\boldsymbol{uvvw}$ by inserting a copy $\boldsymbol{v}$, where the substring $\boldsymbol{vv}$ is called a *(tandem) repeat* with length $2k$. Let $\leqslant k$-TDs represent TDs of length at most k. In this proposal, we focus on $\leqslant$3-TDs, also called *short duplications*. For example, given a string $\boldsymbol{x} = 213012 \in \Sigma_4^*$, a set of $\leqslant$3-TDs may generate an output

$$\boldsymbol{x} = 213012 \to 213\underline{213}012 \to 21321303012 \to 2132\underline{1}303012 = \boldsymbol{x}', \tag{3.1}$$

where the duplicated copies are marked with underlines. Then we define $\boldsymbol{x}'$ as a *descendant* of $\boldsymbol{x}$, i.e., a string generated from $\boldsymbol{x}$ by a set of $\leqslant$3-TDs. Furthermore, for a string $\boldsymbol{x} \in \Sigma_q^*$, let $D^*(\boldsymbol{x})$ be the set of the descendants generated from $\boldsymbol{x}$ by an arbitrary number of $\leqslant$3-TDs.

A *deduplication* of length k is an operation of replacing a repeat $\boldsymbol{vv}$ with $\boldsymbol{v}$ with $|\boldsymbol{v}| = k$. Then $\leqslant$3-deduplications are deduplications with length upper bound by 3. In this chapter, $\leqslant$3-deduplications are simply called *deduplications*. For example, the string $\boldsymbol{x}$ in (3.1) can be recovered from $\boldsymbol{x}'$ by three deduplications.

The set of *$\leqslant$3-irreducible strings* of length n over Σ_q, denoted $\mathrm{Irr}_q(n)$, consists of strings without repeats of the form $\boldsymbol{vv}$, where $|\boldsymbol{v}| \leqslant 3$. Note that $\mathrm{Irr}_q(n)$ is affected by the alphabet size q. For simplicity, $\mathrm{Irr}_q(n)$ may also be denoted as $\mathrm{Irr}(n)$. Let $\mathrm{Irr}(*)$ represent all irreducible strings of finite length. A *duplication root* of $\boldsymbol{x}'$ is a $\leqslant$3-irreducible string $\boldsymbol{x}$ such that $\boldsymbol{x}'$ is a descendant of $\boldsymbol{x}$. Equivalently, $\boldsymbol{x}$ can be obtained from $\boldsymbol{x}'$ by performing all possible deduplications of length at most k. The set of duplication roots of $\boldsymbol{x}'$ is denoted $R(\boldsymbol{x}')$, i.e.,

$$R(\boldsymbol{x}') = \{\boldsymbol{x} \in \mathrm{Irr}(*) | \boldsymbol{x}' \in D^*(\boldsymbol{x})\}.$$

For $\leqslant$3-TDs, the work [30] showed that $R(\boldsymbol{x}')$ has a single element[1]. When $R(\boldsymbol{x}')$ is a singleton, we may treat it as a string instead of a set. The uniqueness of the root implies that if $\boldsymbol{x}''$ is a descendant of $\boldsymbol{x}'$, then $R(\boldsymbol{x}') = R(\boldsymbol{x}'')$.

Besides $\leqslant$3-TDs, we also consider substitution errors, where each substitution replaces an arbitrary symbol by another symbol from the same alphabet. Continuing the example in (3.1), the output after two substitutions and two $\leqslant$3-TDs on $\boldsymbol{x}'$ may be

$$\begin{aligned}\boldsymbol{x}' = 213221303012 &\to 213211303012 \to 21321321\underline{1}1303012\\ &\to 2132132113\underline{2}3012 \to 213213211323\underline{3}\underline{2}3012 = \boldsymbol{x}'',\end{aligned}$$

where the symbols generated from substitutions are marked in red. Let $D^{*,\leqslant p}(\boldsymbol{x})$ represent the set of strings derived from $\boldsymbol{x}$ by an arbitrary number of $\leqslant$3-TDs and at most p substitutions. In the example above, we have $\boldsymbol{x}'' \in D^{*,\leqslant 2}(\boldsymbol{x})$.

Construction 40. *(c.f.[30]) Given $q \geqslant 3$ and a positive integer n, let $\mathcal{C}_n^d = \mathrm{Irr}(n)$ over Σ_q.*

Example 41. *Given $\Sigma_3 = \{0,1,2\}$ and $n = 5$, the code with 30 codewords is*

$$\begin{aligned}\mathcal{C}_5^d = \mathrm{Irr}(5) = \{&01020, 01021, 01201, 01202, 01210, 02010, 02012, 02101, 02102, 02120,\\ &10120, 10121, 10201, 10210, 10212, 12010, 12012, 12021, 12101, 12102,\\ &20102, 20120, 20121, 20210, 20212, 21012, 21020, 21021, 21201, 21202\}.\end{aligned}$$

Lemma 42. *Given $q \geqslant 3$, the code $\mathcal{C}_n^d$ achieves the same asymptotic code rate as the original code $\mathcal{C}_D$ in [30, Construction C] to correct an arbitrary number of short duplications with the highest known asymptotic rate. Furthermore, given $q \geqslant 4$, $\|\mathcal{C}_D\| \leqslant \frac{q-2}{q-3}\|\mathcal{C}_n^d\|$.*

The key notations used in this chapter are summarized in Table 3.1.

3.3 Correcting multiple short duplications and one edit error

In this subsection, we focus on correcting errors that may arise from channels with many duplication errors of length at most 3 and one edit error, which may occur in any position in the string. Note that for duplications whose length is at most 3, the case most relevant to this chapter, Jain et al. [30] proposed error-correcting codes that were shown to have an asymptotically optimal rate by Kovačević [35]. Considering a single edit error reveals important insights into the interactions between edit and duplication errors and will be of use for studying the general case of t edit errors. Based on Example 39, given that an arbitrary number of duplications are possible, an unbounded segment of the output word may be affected by a substitution errors, and, for example, the substituted symbol may appear many times. However, relying on the fact that short tandem duplications lead to regular languages, we show that with an appropriate construction and preprocessing of the

[1]Note that this statement only applies to duplications of length at most 3. For duplications of length at most 4, the root is not unique.

Table 3.1: Key notations in Chapter 3

Notation	Definition
$\Sigma_q = \{0, 1, \cdots, q-1\}$	the alphabet set with q elements
$\|\boldsymbol{u}\|$	the length of a sequence $\boldsymbol{u}$
$\|\|S\|\|$	the number of elements in the set S
$\leqslant k$-TD	a tandem duplication to generate a substring $\boldsymbol{aa}$ from $\boldsymbol{a}$ with $\|\boldsymbol{a}\| \leqslant k$
$\leqslant$ 3-TD/STD	a tandem duplication to generate a substring $\boldsymbol{aa}$ from $\boldsymbol{a}$ with $\|\boldsymbol{a}\| \leqslant 3$
$\mathrm{Irr}_q(n)/\mathrm{Irr}(n)$	the set of all length-n irreducible sequences without tandem repeats $\boldsymbol{aa}$ of length $\leqslant k$
$R(\boldsymbol{x})$	the root of $\boldsymbol{x}$ by removing substrings $\boldsymbol{a}$ from $\boldsymbol{aa}$ with $\|\boldsymbol{a}\| \leqslant 3$.
$\mathrm{Irr}_q(n)/\mathrm{Irr}(n)$	removing all substrings 0^k in $\boldsymbol{z}$
$D^{*,\leqslant p}(\boldsymbol{x})$	the set of all outputs generated from $\boldsymbol{x}$ by many STDs and at most p substitutions
DS(p) channel	an output may suffer from many STDs and at most p substitutions from an input
DE(p) channel	an output may suffer from many STDs and at most p edit errors from an input
DSD(p) channel	an output may suffer from many STDs and at most p substitutions from an input, followed by removing all $\boldsymbol{a}$ from $\boldsymbol{aa}$ with $\|\boldsymbol{a}\| \leqslant k$
DED(p) channel	an output may suffer from many STDs and at most p edit errors from an input, followed by removing all $\boldsymbol{a}$ from $\boldsymbol{aa}$ with $\|\boldsymbol{a}\| \leqslant k$

output of the channel, the deleterious effects of the errors may be localized. We leverage constrained coding and maximum distance separable codes to design codes for correcting the resulting errors, establish a lower bound on the code rate, and provide an asymptotic analysis that shows that the code has rate at least $\log(q-2)$, where q is the size of the alphabet and the log is in base 2. We note that the rate of the code correcting only short duplications is upper bounded by $\log(q-1)$. When $q = 4$, the case corresponding to DNA storage, we provide a computational bound for the code rate, showing that asymptotically its rate is only 0.003 bits/symbol smaller than the code that corrects short duplications but no edit.

We will first consider only substitution edits and construct error-correcting codes capable of correcting many short duplications and a substitution. We will then prove that the same code can correct any number of duplications and an edit error by transforming insertion and deletion errors to substitution errors.

3.3.1 Channels with many $\leqslant$3-TDs and one substitution error

In this subsection, we study channels that alter the input string by applying an arbitrary number of duplication errors and at most one substitution error, where the substitution may occur at any time in the sequence of errors. We will first study the conditions a code must satisfy to be able to

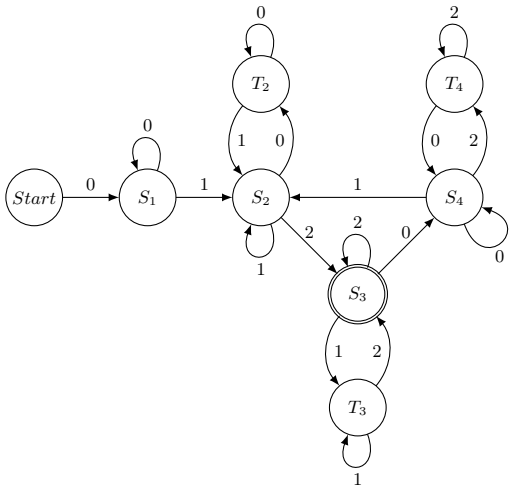


Figure 3.1: Finite automaton for the regular language $D^*(012)$ based on [29].

correct such errors. Then, we will investigate the effect of such channels on the duplication root of sequences, which is an important aspect of designing our error-correcting codes.

A code C is able to correct an arbitrary number of $\leqslant$3-TDs and a substitution if and only if for any two distinct codewords $\boldsymbol{c}_1, \boldsymbol{c}_2 \in C$, we have

$$D^{*,\leqslant 1}(\boldsymbol{c}_1) \cap D^{*,\leqslant 1}(\boldsymbol{c}_2) = \varnothing.$$

To satisfy this condition, it is sufficient to have

$$R(D^{*,\leqslant 1}(\boldsymbol{c}_1)) \cap R(D^{*,\leqslant 1}(\boldsymbol{c}_2)) = \varnothing. \tag{3.2}$$

Condition (3.2) implies that for distinct codewords $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$, $R(\boldsymbol{c}_1) \neq R(\boldsymbol{c}_2)$. Since both $R(\boldsymbol{c}_1)$ and $R(\boldsymbol{c}_2)$ are singletons, this latter condition is in fact sufficient for correcting only $\leqslant$3-TDs since this type of error does not alter the duplication root. For correcting only $\leqslant$3-TDs, defining the code as the set of irreducible strings of a given length leads to asymptotically optimal codes [30], [35]. The decoding process is simply finding the root of the received word.

We take a similar approach to correcting many $\leqslant$3-TDs and a substitution. More specifically, the proposed code C is a subset of $\leqslant$3-irreducible strings, i.e., $R(\boldsymbol{c}) = \boldsymbol{c}$ for $\boldsymbol{c} \in C$. To recover $\boldsymbol{c}$ from the received word $\boldsymbol{y}$, we find $R(\boldsymbol{y})$ and from that recover $R(\boldsymbol{c}) = \boldsymbol{c}$, as will be discussed.

We start by studying the effect of $\leqslant$3-TDs and one substitution on the root of a string. Specifically, for strings $\boldsymbol{x}$ and $\boldsymbol{x}'' \in D^{*,\leqslant 1}(\boldsymbol{x})$, it is of interest to determine how $R(\boldsymbol{x}'')$ differs from $R(\boldsymbol{x})$. We either have $\boldsymbol{x}'' \in D^*(\boldsymbol{x})$, i.e., $\boldsymbol{x}''$ suffers only duplications, or $\boldsymbol{x}'' \in D^{*,1}(\boldsymbol{x})$. In the former case $R(\boldsymbol{x}'') = R(\boldsymbol{x})$. Hence, below we consider only $\boldsymbol{x}'' \in D^{*,1}(\boldsymbol{x})$. Note that duplications that occur after the substitution do not affect the root and so in our analysis we may assume that the substitution is the last error. We start by providing a useful definition and auxiliary lemmas.

Let $\boldsymbol{s}$ and $\bar{\boldsymbol{s}}$ be strings of length n, and let A be the set of symbols in $\boldsymbol{s}$ and $\bar{A}$ the set of symbols

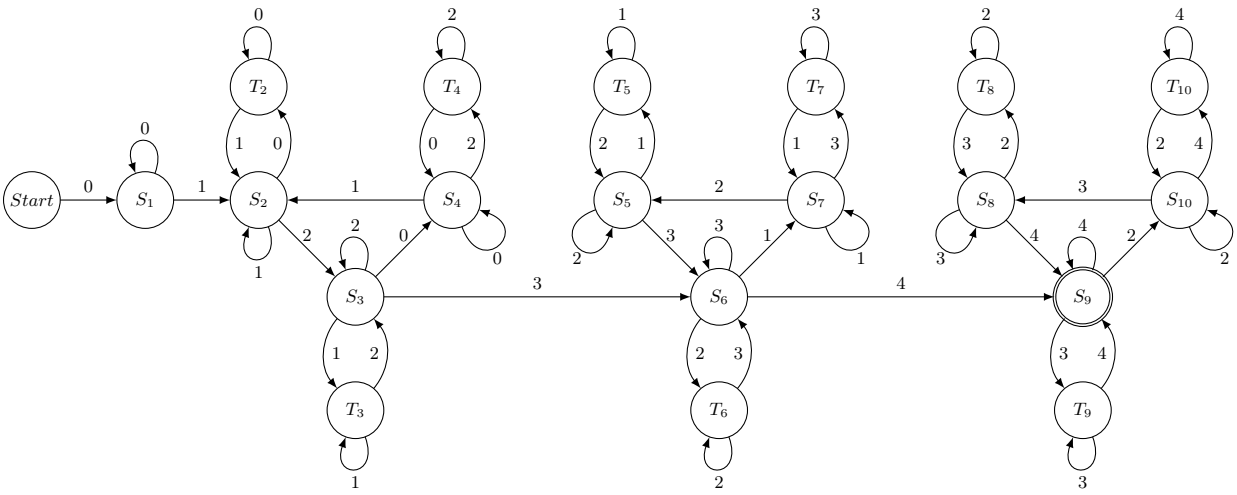


Figure 3.2: Finite automaton for the regular language $D^*(01234)$ based on [29].

in $\bar{\boldsymbol{s}}$. We say that $\boldsymbol{s}$ *dominates* $\bar{\boldsymbol{s}}$ if there exists a function $f : A \to \bar{A}$ such that $\bar{\boldsymbol{s}} = f(\boldsymbol{s})$, where $f(\boldsymbol{s}) = f(s_1) \cdots f(s_n)$. For example, 0102 dominates 1212 (using the mapping $f(0) = 1, f(1) = 2, f(2) = 2$) but 0102 does not dominate 0010. The string $012 \cdots k$ dominates any string of length $k + 1$.

The following lemma helps reduce the number of cases we need to consider by using the dominance relationship defined above.

Lemma 43. *Suppose $\boldsymbol{s}$ dominates $\bar{\boldsymbol{s}}$. The following hold:*

1. *Suppose we apply the same duplication in both $\boldsymbol{s}$ and $\bar{\boldsymbol{s}}$ (that is, in the same position and with the same length). Let the resulting strings be $\boldsymbol{s}'$ and $\bar{\boldsymbol{s}}'$, respectively. Then $\boldsymbol{s}'$ dominates $\bar{\boldsymbol{s}}'$.*

2. *If a deduplication is possible in $\boldsymbol{s}$, a deduplication in the same position and with the same length is possible in $\bar{\boldsymbol{s}}$. Let the result of applying this deduplication to $\boldsymbol{s}$ and $\bar{\boldsymbol{s}}$ be denoted by $\boldsymbol{s}'$ and $\bar{\boldsymbol{s}}'$, respectively. Then $\boldsymbol{s}'$ dominates $\bar{\boldsymbol{s}}'$.*

3. *Let $\bar{\boldsymbol{s}}'$ be obtained from $\bar{\boldsymbol{s}}$ via a substitution in position i and let $\boldsymbol{s}'$ be obtained from $\boldsymbol{s}$ by substituting the symbol in position i with a symbol x not present in $\boldsymbol{s}$. Then, $\boldsymbol{s}'$ dominates $\bar{\boldsymbol{s}}'$.*

4. *We have $|R(\bar{\boldsymbol{s}})| \leqslant |R(\boldsymbol{s})|$.*

Before proving the lemma, below we provide an example for each statement, where duplicated, deduplicated, and substituted symbols are underlined. For i) consider

$$\boldsymbol{s} = 0102 \to \boldsymbol{s}' = 010\underline{10}2$$
$$\bar{\boldsymbol{s}} = 0101 \to \bar{\boldsymbol{s}}' = 010\underline{10}1,$$

for ii) consider

$$\boldsymbol{s} = 0102\underline{10}2\underline{1} \to \boldsymbol{s}' = 01021$$

$$\bar{\boldsymbol{s}} = 0101\underline{101}1 \to \bar{\boldsymbol{s}}' = 01011,$$

for iii) consider

$$\boldsymbol{s} = 01021021 \to \boldsymbol{s}' = 0102\underline{3}021$$
$$\bar{\boldsymbol{s}} = 01011011 \to \bar{\boldsymbol{s}}' = 0101\underline{0}011,$$

and for iv) consider

$$\boldsymbol{s} = 0102\underline{102}1 \to 01021 = R(\boldsymbol{s})$$
$$\bar{\boldsymbol{s}} = 0101\underline{101}1 \to 01\underline{01}1 \to 01\underline{1} \to 01 = R(\bar{\boldsymbol{s}}).$$

Proof. Let f be a function that can show $\boldsymbol{s}$ dominates $\bar{\boldsymbol{s}}$. i) For the first statement, the same mapping f also shows that $\boldsymbol{s}'$ dominates $\bar{\boldsymbol{s}}'$. ii) For the second statement, consider a repeat $\boldsymbol{aa}$ in $\boldsymbol{s}$. Then the repeat $f(\boldsymbol{a})f(\boldsymbol{a})$ is present in $\bar{\boldsymbol{s}}$ in the same position. So the deduplication is possible in $\bar{\boldsymbol{s}}$. The same mapping f proves that $\boldsymbol{s}'$ dominates $\bar{\boldsymbol{s}}'$. iii) For the third statement, let the substitution in $\bar{\boldsymbol{s}}$ alter the symbol in position i to some symbol a. If we extend f by mapping x to a, then f proves that $\boldsymbol{s}'$ dominates $\bar{\boldsymbol{s}}'$. iv) From ii), any sequence of deduplications applied to $\boldsymbol{s}$ can also be applied to $\bar{\boldsymbol{s}} = f(\boldsymbol{s})$. In particular, the sequence of deduplications that takes $\boldsymbol{s}$ to its root $R(\boldsymbol{s})$ takes $\bar{\boldsymbol{s}} = f(\boldsymbol{s})$ to $f(R(\boldsymbol{s}))$. The root of $\bar{\boldsymbol{s}}$ can be obtained by removing any remaining repeats in $f(R(\boldsymbol{s}))$ (recall that the root is unique so all sequences of deduplications must lead to the same sequence). Hence $|R(\bar{\boldsymbol{s}})| \leqslant |f(R(\boldsymbol{s}))|$. Noting $|f(R(\boldsymbol{s}))| = |R(\boldsymbol{s})|$ completes the proof. □

The next lemma studies the length of the roots of the descendants for a special subset of strings.

Lemma 44. *For any alphabet* Σ_q*,*

$$\max_{\boldsymbol{x}\in\Sigma_q^3} \max_{\boldsymbol{x}''\in D^{*,1}(\boldsymbol{x})} |R(\boldsymbol{x}'')| = 13,$$
$$\max_{\boldsymbol{x}\in\Sigma_q^5} \max_{\boldsymbol{x}''\in D^{*,1}(\boldsymbol{x})} |R(\boldsymbol{x}'')| \leqslant 17.$$

Proof. For the first statement, it suffices to consider only $\boldsymbol{x} = 012$ and assume that the substitution that leads to $\boldsymbol{x}''$ replaces a symbol in $\boldsymbol{x}$ with some symbol other than 0, 1, and 2, e.g., 3. To see this, consider any string $\bar{\boldsymbol{x}}$ of length 3 over any alphabet. The string $\bar{\boldsymbol{x}}$ is dominated by $\boldsymbol{x}$. Now consider any $\bar{\boldsymbol{x}}'' \in D^{*,1}(\bar{\boldsymbol{x}})$. There is a sequence of "errors" consisting of duplications, a substitution, and more duplications that transforms $\bar{\boldsymbol{x}}$ to $\bar{\boldsymbol{x}}''$. By Lemma 43, i) and iii), there is a corresponding sequence of errors, consisting of duplications, a substitution, and duplications, that when applied to $\boldsymbol{x}$ will result in $\boldsymbol{x}''$, where $\boldsymbol{x}''$ dominates $\bar{\boldsymbol{x}}''$ (the substitution in the sequence of errors for $\boldsymbol{x}$ substitutes the existing symbol with a symbol not in the set $\{0, 1, 2\}$). Then by Lemma 43, iv), we have $|R(\bar{\boldsymbol{x}}'')| \leqslant |R(\boldsymbol{x}'')|$. Since this is true for any choice of $\bar{\boldsymbol{x}}$ and any $\bar{\boldsymbol{x}}'' \in D^{*,1}(\bar{\boldsymbol{x}})$, it suffices to find

$$\max_{\boldsymbol{x}''\in D^{*,1}(012)} |R(\boldsymbol{x}'')|,$$

Table 3.2: Paths representing irreducible strings starting from and ending at specific states.

State	Irreducible paths from 'Start' to state	Irreducible paths from State to S_3
S_1	0	012, 1012, 12, 12012,
S_2	01, 01201	012,1012, 12, 12012, 2, 2012, 212, 212012
S_3	012	012, 02012, 12, 12012, 2, 2012, 212, 212012
S_4	0120	012, 02012, 1012, 12, 12012, 2012
T_2	010, 012010	012, 1012,12, 12012
T_3	0121	12, 12012, 2, 2012, 212, 212012
T_4	01202	012, 02012, 2012

where the substitution resulting in $\boldsymbol{x}''$ replaces the existing symbol with a symbol not present in $\boldsymbol{x} = 012$. Henceforth, we assume $\boldsymbol{x} = 012$.

As shown in [29], $D^*(\boldsymbol{x})$ is a regular language whose words can be described as paths from 'Start' to S_3 in the finite automaton given in Figure 3.1, where the word associated with each path is the sequence of the edge labels. Let $\boldsymbol{x}' \in D^*(\boldsymbol{x})$ and $\boldsymbol{x}'' \in D^{0,1}(\boldsymbol{x}')$. Assume $\boldsymbol{x}' = \boldsymbol{u}w\boldsymbol{z}$ and $\boldsymbol{x}'' = \boldsymbol{u}\hat{w}\boldsymbol{z}$, where $\boldsymbol{u}, \boldsymbol{z}$ are strings and w and $\hat{w} \notin \{0, 1, 2\}$ are distinct symbols. The string $\boldsymbol{u}$ represents a path from 'Start' to some state U and the string $\boldsymbol{z}$ represents a path from some state Z to S_3 in the automaton, where there is an edge with label w from U to Z.

Since $R(\boldsymbol{x}'') = R(R(\boldsymbol{u})\hat{w}R(\boldsymbol{z}))$, we have $|R(\boldsymbol{x}'')| \leqslant |R(\boldsymbol{u})| + 1 + |R(\boldsymbol{z})|$ (recall that $R(\boldsymbol{s})$ is a singleton for a string $\boldsymbol{s}$). The maximum value for $|R(\boldsymbol{u})|$ is the length of some path from 'Start' to U such that the corresponding sequence does not have any repeats (henceforth, called an *irreducible path*). All such paths/sequences are listed in the second column of Table 3.2 for all choices of U. Similarly, the maximum value for $|R(\boldsymbol{z})|$ is the length of some irreducible path from Z to S_3; all such possibilities are listed in the third column of Table 3.2. An inspection of Table 3.2 shows that choosing $U = T_2$ and $Z = S_2$ leads to the largest value of $|R(\boldsymbol{u})| + 1 + |R(\boldsymbol{z})|$, namely $6 + 1 + 6 = 13$. We note that the specific sequence achieving this length is $\boldsymbol{x}'' = 0120103212012$, which can be obtained via the sequence $\boldsymbol{x} \to 012\,\underline{012}\,\underline{012} \to 012\,01\underline{012}\,012 \to 012\,0101\underline{212}\,012 \to 0120103212012 = \boldsymbol{x}''$ with a substitution $1 \to 3$ in the last step, where we have combined non-overlapping duplications into a single step.

Let us now prove the second statement. Again we need only consider $\boldsymbol{x} = 01234$, for which $D^*(\boldsymbol{x})$ is the regular language whose automaton is shown in Figure 3.2. In a similar manner to the proof of the previous part, we can show that the length of the longest irreducible path from 'Start' to any state in the automaton is at most 8 and the length of the longest irreducible path from any state to S_9 is also at most 8. Hence, $|R(\boldsymbol{x}'')| \leqslant 8 + 1 + 8 = 17$, completing the proof. □

We now consider changes to the roots of arbitrary strings when passed through a channel with arbitrarily many $\leqslant$3-TDs and one substitution. The next lemma is used in the main result of this

section, Theorem 47, which shows that even though a substituted symbol may be duplicated many times, the effect of a substitution on the root is bounded.

Lemma 45. *Let $\boldsymbol{x}$ be any string of length at least 5 and $\boldsymbol{x}' \in D^*(\boldsymbol{x})$. For any decomposition of $\boldsymbol{x}$ as*

$$\boldsymbol{x} = \boldsymbol{r}\ ab\ \boldsymbol{t}\ de\ \boldsymbol{s},$$

for $a, b, d, e \in \Sigma_q$ and $\boldsymbol{r}, \boldsymbol{t}, \boldsymbol{s} \in \Sigma_q^*$, with $\boldsymbol{t}$ nonempty, there is a decomposition of $\boldsymbol{x}'$ as

$$\boldsymbol{x}' = \boldsymbol{u}\ ab\ \boldsymbol{w}\ de\ \boldsymbol{v}$$

such that $\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{v} \in \Sigma_q^*$, $\boldsymbol{u}ab \in D^*(\boldsymbol{r}ab)$, $ab\boldsymbol{w}de \in D^*(ab\boldsymbol{t}de)$, and $de\boldsymbol{v} \in D^*(de\boldsymbol{s})$.

Proof. If $\boldsymbol{x} = \boldsymbol{x}'$, the claim is true since we may choose $\boldsymbol{u} = \boldsymbol{r}, \boldsymbol{w} = \boldsymbol{t}, \boldsymbol{v} = \boldsymbol{s}$. It suffices to consider the case in which $\boldsymbol{x}'$ is obtained from $\boldsymbol{x}$ via a single duplication. The case of more duplications can be proved inductively.

First suppose the length of the duplication transforming $\boldsymbol{x}$ to $\boldsymbol{x}'$ is 1. If this duplication occurs in $\boldsymbol{r}$, we choose $\boldsymbol{u}$ to be the descendant of $\boldsymbol{r}$ and let $\boldsymbol{w} = \boldsymbol{t}$ and $\boldsymbol{v} = \boldsymbol{s}$, satisfying the claim. Duplication of a single symbol in $\boldsymbol{t}$ or $\boldsymbol{s}$ is handled similarly. If a is duplicated, we let $\boldsymbol{u} = \boldsymbol{r}a$, $\boldsymbol{w} = \boldsymbol{t}$, $\boldsymbol{v} = \boldsymbol{s}$. If b is duplicated, we let $\boldsymbol{u} = \boldsymbol{r}$, $\boldsymbol{w} = b\boldsymbol{t}$, $\boldsymbol{v} = \boldsymbol{s}$. The cases for d and e are similar.

Second, consider a duplication of length 2 or 3. Such a duplication is fully contained in $\boldsymbol{r}ab$, $ab\boldsymbol{t}de$, or $de\boldsymbol{s}$. A duplication of length 2 or 3 applied to a string $\boldsymbol{z}$ does not alter the first two and the last two symbols of $\boldsymbol{z}$. So, for example, if the duplication occurs in $\boldsymbol{r}ab$, then we can choose $\boldsymbol{u}$ such that $\boldsymbol{u}ab \in D^1(\boldsymbol{r}ab)$ and let $\boldsymbol{w} = \boldsymbol{t}$ and $\boldsymbol{v} = \boldsymbol{s}$. The cases of duplications contained in the other strings are similar. □

We now provide an example in which we illustrate how the root of a string can be altered by several duplications and one substitution.

Example 46. *Fix $\Sigma_4 = \{0, 1, 2, 3\}$ as the alphabet. In the following examples, $\boldsymbol{x}$ is an irreducible string, $\boldsymbol{x}' \in D^*(\boldsymbol{x})$, and $\boldsymbol{x}'' \in D^{0,1}(\boldsymbol{x}')$. We compare $R(\boldsymbol{x}) = \boldsymbol{x}$ with $R(\boldsymbol{x}'')$. In particular, we will decompose $R(\boldsymbol{x})$ and $R(\boldsymbol{x}'')$ as $R(\boldsymbol{x}) = \boldsymbol{\alpha\beta\gamma}$ and $R(\boldsymbol{x}'') = \boldsymbol{\alpha\beta'\gamma}$. In other words, $R(\boldsymbol{x}'')$ can be obtained from $R(\boldsymbol{x})$ by deleting $\boldsymbol{\beta}$ and inserting $\boldsymbol{\beta}'$.*

- *Let $\boldsymbol{x} = 012302$, $\boldsymbol{x}' = 011201201230202$, and $\boldsymbol{x}'' = 011201301230202$, where the underlined symbols result from duplication and the bold symbol from substitution. Then $R(\boldsymbol{x}'') =$ 012013012302 and the change from $R(\boldsymbol{x})$ to $R(\boldsymbol{x}'')$ can be viewed as*

$$R(\boldsymbol{x}) = \underbrace{012}_{\boldsymbol{\alpha}}\underbrace{302}_{\boldsymbol{\gamma}} \to R(\boldsymbol{x}'') = \underbrace{012}_{\boldsymbol{\alpha}}\underbrace{013012}_{\boldsymbol{\beta'}}\underbrace{302}_{\boldsymbol{\gamma}},$$

  with $\boldsymbol{\beta} = \Lambda$.

- *Let $\boldsymbol{x} = 13203103$, $\boldsymbol{x}' = 1313213203103103$, and $\boldsymbol{x}'' = 1313213103103103$. Then $R(\boldsymbol{x}'') =$*

13213103 *and the change from* $R(\boldsymbol{x})$ *to* $R(\boldsymbol{x}'')$ *can be viewed as*

$$R(\boldsymbol{x}) = \underbrace{132}_{\boldsymbol{\alpha}}\underbrace{0}_{\boldsymbol{\beta}}\underbrace{3103}_{\boldsymbol{\gamma}} \to R(\boldsymbol{x}'') = \underbrace{132}_{\boldsymbol{\alpha}}\underbrace{1}_{\boldsymbol{\beta}'}\underbrace{3103}_{\boldsymbol{\gamma}}.$$

- *Let* $\boldsymbol{x} = 012010321201230$, $\boldsymbol{x}' = 012\underline{012}0103212\underline{0}\underline{2}01\underline{201}230$, *and* $\boldsymbol{x}'' = 012\underline{012}0101212\underline{0}\underline{2}01\underline{201}230$. *Then* $R(\boldsymbol{x}'') = 01230$ *and the change from* $R(\boldsymbol{x})$ *to* $R(\boldsymbol{x}'')$ *can be viewed as*

$$R(\boldsymbol{x}) = \underbrace{012}_{\boldsymbol{\alpha}}\underbrace{0103212012}_{\boldsymbol{\beta}}\underbrace{30}_{\boldsymbol{\gamma}} \to R(\boldsymbol{x}'') = \underbrace{012}_{\boldsymbol{\alpha}}\underbrace{30}_{\boldsymbol{\gamma}}, \tag{3.3}$$

  with $\boldsymbol{\beta}' = \Lambda$.

Let $\mathcal{L}$ be the smallest integer, if it exists, such that for any alphabet Σ_q, any $\boldsymbol{x} \in \Sigma_q^*$, and any $\boldsymbol{x}'' \in D^{*,1}(\boldsymbol{x})$, we can obtain $R(\boldsymbol{x}'')$ from $R(\boldsymbol{x})$ by deleting a substring of length at most $\mathcal{L}$ and inserting a substring of length at most $\mathcal{L}$ in the same position. The example given in (3.3) shows that $\mathcal{L}$, if it exists, satisfies $\mathcal{L} \geqslant 10$. We note however that the definition does not guarantee that $\mathcal{L}$ exists as we may be able to produce examples in which the length of the deleted or the inserted substring is arbitrarily long. The next theorem shows that such examples cannot be constructed by providing an explicit upper bound on $\mathcal{L}$.

Theorem 47. $\mathcal{L}$ *exists (i.e., it is finite). Moreover,* $\mathcal{L} \leqslant 17$.

Proof. We may assume $\boldsymbol{x}$ is irreducible. If it is not, let $\boldsymbol{x}_0 = R(\boldsymbol{x})$ so that $\boldsymbol{x}'' \in D^{*,1}(\boldsymbol{x}) \subseteq D^{*,1}(\boldsymbol{x}_0)$. If the statement of the theorem holds for $\boldsymbol{x}_0$, it also holds for $\boldsymbol{x}$ since $R(\boldsymbol{x}) = R(\boldsymbol{x}_0)$.

We will find $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\beta}', \boldsymbol{\gamma} \in \Sigma_q^*$ with $R(\boldsymbol{x}) = \boldsymbol{\alpha\beta\gamma}$ and $R(\boldsymbol{x}'') = \boldsymbol{\alpha\beta'\gamma}$ such that $|\boldsymbol{\beta}'| \leqslant 17$. By symmetry, it suffices to prove $|\boldsymbol{\beta}'| \leqslant 17$ for all irreducible $\boldsymbol{x}$. To see this symmetry, note that $\boldsymbol{\alpha\beta'\gamma}$ is obtained from $\boldsymbol{\alpha\beta\gamma}$ by applying, in order, duplications, a single substitution, more duplications, and finally removing all repeats (performing all possible deduplications). Recall that for $\leqslant$3-TDs, the root is unique and regardless of the order in which deduplications are applied, we will arrive at the same root. In other words, applying a sequence of duplications to a string $\boldsymbol{s}$ and then removing all repeats is equivalent to removing all repeats from $\boldsymbol{s}$. Hence, we may instead assume that the process transforming $\boldsymbol{\alpha\beta\gamma}$ to $\boldsymbol{\alpha\beta'\gamma}$ is as follows: duplications, substitution, deduplications. Since this process is reversible, general statements that hold for $\boldsymbol{\beta}'$ also hold for $\boldsymbol{\beta}$.

Let $\boldsymbol{x}' \in D^*(\boldsymbol{x})$ be obtained from $\boldsymbol{x}$ through duplications and $\boldsymbol{x}''$ be obtained from $\boldsymbol{x}'$ through a substitution. We assume that $\boldsymbol{x} = \boldsymbol{r}abcde\boldsymbol{s}$, where $\boldsymbol{r}, \boldsymbol{s} \in \Sigma_q^*$ and $a, b, c, d, e \in \Sigma_q$, such that the substituted symbol in $\boldsymbol{x}'$ is a copy of c. Note that if $|\boldsymbol{x}| < 5$ or if a copy of one of its first two symbols or its last two symbols are substituted, then we can no longer write $\boldsymbol{x}$ as described. To avoid considering these cases separately, we may append two dummy symbols to the beginning of $\boldsymbol{x}$ and two dummy symbols to the end of $\boldsymbol{x}$, where the four dummy symbols are distinct and do not belong to Σ_q, and prove the result for this new string. Since these dummy symbols do not participate in any duplication, substitution, or deduplication events, the proof is also valid for the original $\boldsymbol{x}$.

With the above assumption and based on Lemma 45, we can write

$$\begin{aligned} \boldsymbol{x} &= \ \boldsymbol{r}\ ab\ c\ de\ \boldsymbol{s} \\ \boldsymbol{x}' &= \boldsymbol{u}\ ab\ \boldsymbol{w}\ de\ \boldsymbol{v} \in D^*(\boldsymbol{x}), \\ \boldsymbol{x}'' &= \boldsymbol{u}\ ab\ \boldsymbol{z}\ de\ \boldsymbol{v} \in D^{0,1}(\boldsymbol{x}'), \end{aligned} \tag{3.4}$$

where $\boldsymbol{u}ab \in D^*(\boldsymbol{r}ab)$, $ab\boldsymbol{w}de \in D^*(abcde)$, $de\boldsymbol{v} \in D^*(de\boldsymbol{s})$, and $\boldsymbol{z}$ is obtained from $\boldsymbol{w}$ by substituting an occurrence of c. From (3.12), $R(\boldsymbol{x}'') = R(\boldsymbol{r}R(ab\boldsymbol{z}de)\boldsymbol{s})$, where $R(ab\boldsymbol{z}de)$ starts with ab and ends with de (which may fully or partially overlap). The outer R in $R(\boldsymbol{r}R(ab\boldsymbol{z}de)\boldsymbol{s})$ may remove some symbols at the end of $\boldsymbol{r}$, beginning and end of $R(ab\boldsymbol{z}de)$, and the beginning of $\boldsymbol{s}$, leading to $\boldsymbol{\alpha}\boldsymbol{\beta}'\boldsymbol{\gamma}$, where $\boldsymbol{\alpha}$ is a prefix of $\boldsymbol{r}$, $\boldsymbol{\beta}'$ is a substring of $R(ab\boldsymbol{z}de)$, and $\boldsymbol{\gamma}$ is a suffix of $\boldsymbol{s}$. Hence, $|\boldsymbol{\beta}'| \leqslant |R(ab\boldsymbol{z}de)|$. But $ab\boldsymbol{z}de \in D^{*,1}(abcde)$ and thus by Lemma 44, $|R(ab\boldsymbol{z}de)| \leqslant 17$, completing the proof. □

3.3.2 Error-correcting codes

Having studied how duplication roots are affected by tandem duplication and substitution errors, in Subsection 3.3.2, we construct codes that can correct such errors. In Subsection 3.3.3, we show that the same codes can correct duplication and edit errors. We will also determine the rate of these codes and compare it with the rate of codes that only correct duplications, which provides an upper bound.

As noted in the previous section, the effect of a substitution error on the root of the stored codeword is local in the sense that a substring of bounded length may be deleted and another substring of bounded length may be inserted in its position. A natural approach to correcting such errors is to divide the codewords into blocks such that this alteration can affect a limited number of blocks. In particular, we divide the string into *message blocks* that are separated by *marker blocks* known to the decoder. We start with an auxiliary construction.

Construction 48. *Let l, m, N be positive integers with $m > l$ and $\boldsymbol{\sigma} \in \mathrm{Irr}(l)$. The code $\mathcal{C}_{\boldsymbol{\sigma}}$ of length $n = N(m+l) - l$ over Σ_q consists of irreducible strings $\boldsymbol{x}$ obtained by alternating between* message blocks *of length m and copies of the* marker *sequence $\boldsymbol{\sigma}$, i.e.,*

$$\boldsymbol{x} = B_1\boldsymbol{\sigma}B_2\boldsymbol{\sigma}\cdots\boldsymbol{\sigma}B_N,$$

such that $\boldsymbol{x} \in \mathrm{Irr}(N(m+l)-l)$, $B_i \in \mathrm{Irr}(m) \subseteq \Sigma_q^m, i \in [N]$, and there are exactly two occurrences of $\boldsymbol{\sigma}$ in $\boldsymbol{\sigma}B_i\boldsymbol{\sigma}$, for all $i \in [N]$. (Thus, there are precisely $N-1$ occurrences of $\boldsymbol{\sigma}$ in $\boldsymbol{x}$.)

We remark that for our purposes, we can relax the condition on $\boldsymbol{\sigma}B_i\boldsymbol{\sigma}$ for $i = 1, N$. Specifically, it suffices to have exactly one occurrence of $\boldsymbol{\sigma}$ in $B_1\boldsymbol{\sigma}$ and one occurrence of $\boldsymbol{\sigma}$ in $\boldsymbol{\sigma}B_N$. For simplicity however, we do not use these relaxed conditions.

Example 49. *Let $m = 6$, $N = 5$, and $\boldsymbol{\sigma} = 01231$ with $l = 5$. Then the code $\mathcal{C}_{\boldsymbol{\sigma}}$ in Construction 48*

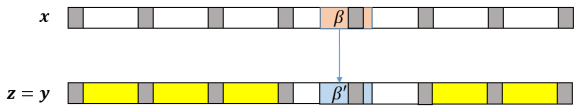


Figure 3.3: If marker sequences, shown as gray, are in the same positions in the codeword $\boldsymbol{x}$ and the retrieved string $\boldsymbol{y}$, then $\boldsymbol{\beta}$ and $\boldsymbol{\beta}'$ have the same length and at most two of the message blocks are affected by the errors, as discussed in the proof of Theorem 50.

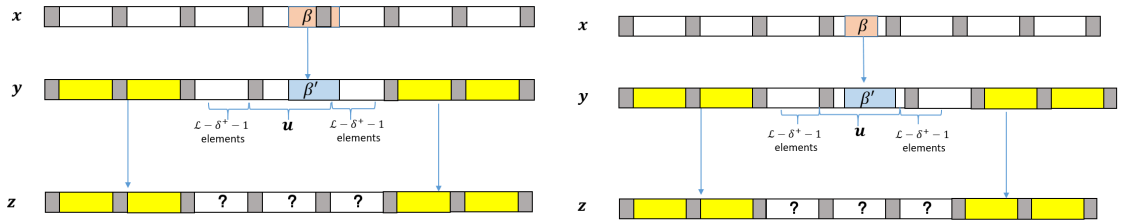


Figure 3.4: If marker sequences, shown as gray, are in different positions in the codeword $\boldsymbol{x}$ and the retrieved string $\boldsymbol{y}$, then a substring $\boldsymbol{u}$ is identified and then expanded to ensure it contains $\boldsymbol{\beta}'$. Those blocks in $\boldsymbol{y}$ that intersect with this expanded substring are marked as erasures while other blocks are error-free message blocks, as described in the proof of Theorem 50.

will contain the codeword

$$\boldsymbol{x} = 012013012310301210123120231201231303203012312030023,$$

where the message blocks $B_1 = 012013$, $B_2 = 030121$, $B_3 = 202312$, $B_4 = 303203$, $B_5 = 203023$ *are marked in red. Furthermore,* $\boldsymbol{\sigma} B_i \boldsymbol{\sigma} \in \mathrm{Irr}(16)$ *for* $i \in [5]$*, which, as will be shown in Lemma 51 below, implies that the codeword* $\boldsymbol{x} \in \Sigma_4^{50}$ *is irreducible.*

Given an input $\boldsymbol{x}$ with N message blocks, let $\boldsymbol{y}$ be the root of the output after duplications and at most one substitution. We define a *block* in $\boldsymbol{y}$ as a maximal substring that does not overlap with any separator sequence $\boldsymbol{\sigma}$. Note that a block in $\boldsymbol{y}$ may or may not be an error-free message block from $\boldsymbol{x}$.

With Construction 48 in hand, in the next theorem, we show that the effect of one substitution and many tandem duplications is limited to a small number of message blocks. We note that $\mathcal{L}$ appearing in the theorem below was defined before Theorem 47 and satisfies $\mathcal{L} \leqslant 17$.

Theorem 50. *Let* $\mathcal{C}_{\boldsymbol{\sigma}}$ *be the code defined in Construction 48. If* $m > \mathcal{L}$*, then there exists a decoder* $\mathcal{D}_{\boldsymbol{\sigma}}$ *that, for any* $\boldsymbol{x} \in \mathcal{C}_{\boldsymbol{\sigma}}$ *and* $\boldsymbol{y} \in R(D^{*,\leqslant 1}(\boldsymbol{x}))$*, outputs* $\boldsymbol{z} = \mathcal{D}_{\boldsymbol{\sigma}}(\boldsymbol{y})$ *such that, relative to* $\boldsymbol{x}$*, either two of the message blocks* B_i *are substituted in* $\boldsymbol{z}$ *or four of them are erased.*

Proof. Let $\boldsymbol{x} = \boldsymbol{\alpha}\boldsymbol{\beta}\boldsymbol{\gamma}$ and $\boldsymbol{y} = \boldsymbol{\alpha}\boldsymbol{\beta}'\boldsymbol{\gamma}$, where by Theorem 47, $|\boldsymbol{\beta}|, |\boldsymbol{\beta}'| \leqslant \mathcal{L}$. To avoid a separate treatment for blocks B_1 and B_N, the decoder appends $\boldsymbol{\sigma}$ to the beginning and the end of $\boldsymbol{y}$ and

assumes that the codewords are of the form $\boldsymbol{\sigma}B_1\boldsymbol{\sigma}\cdots\boldsymbol{\sigma}B_N\boldsymbol{\sigma}$. The decoder considers two cases depending on whether the marker sequences $\boldsymbol{\sigma}$ are in the same positions in $\boldsymbol{y}$ as in the codewords in $\mathcal{C}_{\boldsymbol{\sigma}}$. If the markers are in the same positions, as shown in Figure 3.3, then $|\boldsymbol{x}| = |\boldsymbol{y}|$, and consequently, $|\boldsymbol{\beta}| = |\boldsymbol{\beta}'| \leqslant \mathcal{L}$. Since $\mathcal{L} < m = |B_i|$, at most two (adjacent) blocks B_i are affected by substituting $\boldsymbol{\beta}$ by $\boldsymbol{\beta}'$ and thus $\boldsymbol{z} = \boldsymbol{y}$ differs from $\boldsymbol{x}$ in at most two blocks.

On the other hand, if the markers are in different positions in $\boldsymbol{y}$ compared to the codewords in $\mathcal{C}_{\boldsymbol{\sigma}}$, the decoder uses the location of the markers to identify the position of the message blocks that may be affected and erases them, as described below. By the definition of blocks in $\boldsymbol{y}$, since the markers are in different positions in $\boldsymbol{x}$ and $\boldsymbol{y}$, there is at least one block B in $\boldsymbol{y}$ whose length differs from m. Hence, $\boldsymbol{y}$ has a substring $\boldsymbol{u}$ of length $m + 2l$ that starts with $\boldsymbol{\sigma}$ and contains part or all of B but does not end with $\boldsymbol{\sigma}$. Two examples where such a situation may arise are shown in Figure 3.4. On the left, $|\boldsymbol{\beta}| = |\boldsymbol{\beta}'|$ and a marker is absent from $\boldsymbol{y}$ due to substituting $\boldsymbol{\beta}$ with $\boldsymbol{\beta}'$. On the right, $\boldsymbol{\beta}$ and $\boldsymbol{\beta}'$ have different lengths, and this causes the markers to move.

Let $\delta = |\boldsymbol{x}| - |\boldsymbol{y}| = |\boldsymbol{\beta}| - |\boldsymbol{\beta}'|$ and $\delta^+ = \max(0, \delta)$. Note that $|\boldsymbol{\beta}'| = |\boldsymbol{\beta}| - \delta \leqslant \mathcal{L} - \delta$. Furthermore, $|\boldsymbol{\beta}'| \leqslant \mathcal{L}$ and so $|\boldsymbol{\beta}'| \leqslant \min(\mathcal{L}, \mathcal{L} - \delta) = \mathcal{L} - \delta^+$. Let $\boldsymbol{y}'$ be obtained by removing $\boldsymbol{u}$ along with $\mathcal{L} - \delta^+ - 1$ elements from each of its sides from $\boldsymbol{y}$. This removes $\boldsymbol{\beta}'$ from $\boldsymbol{y}$. More formally, we claim $\boldsymbol{y}'$ can be obtained via a deletion from $\boldsymbol{x}$. First, suppose $|\boldsymbol{\beta}'| = 0$. Then $\boldsymbol{y} = \boldsymbol{\alpha}\boldsymbol{\gamma}$. Note that $\boldsymbol{u}$ is not a substring of $\boldsymbol{x}$ since every substring of $\boldsymbol{x}$ that has length $m + 2l$ and starts with $\boldsymbol{\sigma}$ also ends with $\boldsymbol{\sigma}$. Hence, it must overlap with both $\boldsymbol{\alpha}$ and $\boldsymbol{\gamma}$. After deleting $\boldsymbol{u}$ from $\boldsymbol{y}$, we obtain a string $\boldsymbol{y}' = \boldsymbol{\alpha}'\boldsymbol{\gamma}'$ such that $\boldsymbol{\alpha}'$ is a prefix of $\boldsymbol{\alpha}$ and $\boldsymbol{\gamma}'$ is a suffix of $\boldsymbol{\gamma}$, proving the claim. Next, suppose that $|\boldsymbol{\beta}'| > 0$ and recall that $\boldsymbol{y} = \boldsymbol{\alpha}\boldsymbol{\beta}'\boldsymbol{\gamma}$. Since $\boldsymbol{u}$ is not a substring of $\boldsymbol{x}$, it is not a substring of $\boldsymbol{\alpha}$ or $\boldsymbol{\gamma}$. Furthermore, as $\boldsymbol{u}$ is longer than $\boldsymbol{\beta}'$, it not a substring of $\boldsymbol{\beta}'$ either. Hence, at least one of the following holds: i) $\boldsymbol{u}$ overlaps with both $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}'$ or ii) $\boldsymbol{u}$ overlaps with both $\boldsymbol{\beta}'$ and $\boldsymbol{\gamma}$. Case i) is shown in Figure 3.4. In either case, the substring of $\boldsymbol{y}$ consisting of $\boldsymbol{u}$ and the $\mathcal{L} - \delta^+ - 1$ elements on each side of $\boldsymbol{u}$ contains $\boldsymbol{\beta}'$, proving the claim. Hence, $\boldsymbol{y}'$ is a sequence that relative to $\boldsymbol{x}$ suffers a deletion of length at most $m + 2l + 2\mathcal{L} - 2\delta^+ - 2 + |\boldsymbol{\beta}| - |\boldsymbol{\beta}'| < 3m + 2l$ from a known position. The deletion affects at most 4 message blocks and since its location is known, the decoder can mark these message blocks as erased. □

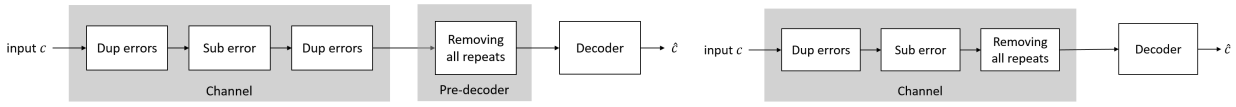
In Construction 48, the constraint that $\boldsymbol{x}$ must be irreducible creates interdependence between the message blocks, making the code more complex. The following lemma allows us to treat each message block independently provided that $\boldsymbol{\sigma}$ is sufficiently long.

Lemma 51. *Let $\boldsymbol{x}$ be as defined in Construction 48 and assume $l \geqslant 5$. The condition $\boldsymbol{x} \in \mathrm{Irr}(N(m+l) - l)$ is satisfied if*

$$\boldsymbol{\sigma}B_i\boldsymbol{\sigma} \in \mathrm{Irr}(m + 2l), \quad \text{for all } i \in [N]. \tag{3.5}$$

Proof. Suppose that $\boldsymbol{x}$ has a repeat $\boldsymbol{a}\boldsymbol{a}$, with $|\boldsymbol{a}| \leqslant 3$. Since $|\boldsymbol{a}\boldsymbol{a}| \leqslant 6$ and $|\boldsymbol{\sigma}| \geqslant 5$, there is no i such that the repeat lies in $B_i\boldsymbol{\sigma}B_{i+1}$ and overlaps both B_i and B_{i+1}. So it must be fully contained in $B_1\boldsymbol{\sigma}$, $\boldsymbol{\sigma}B_N$, or $\boldsymbol{\sigma}B_i\boldsymbol{\sigma}$ for some $2 \leqslant i \leqslant N - 1$, contradicting assumption (3.5). □

We now present a code based on Construction 48 and prove that it can correct any number of tandem duplications and one substitution error.



(i) (ii)

Figure 3.5: The duplication-substitution channel along with the decoder (i) and an equivalent representation of the end-to-end system (ii).

Construction 52. *Let l, m be positive integers with $m > l \geqslant 5$, and $\boldsymbol{\sigma} \in \mathrm{Irr}(l)$. Furthermore, let $\mathcal{B}^m_{\boldsymbol{\sigma}}$ denote the set of sequences B such that $\boldsymbol{\sigma} B \boldsymbol{\sigma} \in \mathrm{Irr}(m + 2l)$ has exactly two occurrences of $\boldsymbol{\sigma}$, and $M = M^{(m)}_{\boldsymbol{\sigma}} = |\mathcal{B}^m_{\boldsymbol{\sigma}}|$. Finally, let t be a positive integer such that $2^t \leqslant M$ and $\zeta : \mathbb{F}_{2^t} \to \mathcal{B}^m_{\boldsymbol{\sigma}}$ be an injective mapping. We define $\mathcal{C}_{MDS}$ as*

$$\mathcal{C}_{MDS} = \{\zeta(c_1)\boldsymbol{\sigma}\zeta(c_2)\boldsymbol{\sigma}\cdots\boldsymbol{\sigma}\zeta(c_N) : \boldsymbol{c} \in \mathrm{MDS}(N, N-4, 5)\},$$

where $\mathrm{MDS}(N, N-4, 5)$ denotes an MDS code over $\mathbb{F}_{2^t}$ of length $N = 2^t - 1$, dimension $N - 4$, and Hamming distance $d_H = 5$.

Note that the mapping ζ exists because $\|\mathbb{F}_{2^t}\| \leqslant \|\mathcal{B}^m_{\boldsymbol{\sigma}}\|$ by the choice of t. For example, we can sort the elements of $\mathbb{F}_{2^t}$ and $\mathcal{B}^m_{\boldsymbol{\sigma}}$ lexicographically and map the ith element of $\mathbb{F}_{2^t}$ to the ith element of $\mathcal{B}^m_{\boldsymbol{\sigma}}$.

Theorem 53. *If $m > \mathcal{L}$, then the error-correcting code $\mathcal{C}_{\mathrm{MDS}}$ in Construction 52 can correct any number of $\leqslant$3-TDs and at most one substitution error.*

Proof. Let the stored codeword be $\boldsymbol{x} = B_1\boldsymbol{\sigma}\cdots\boldsymbol{\sigma}B_N \in \mathcal{C}_{MDS}$, where $B_i = \zeta(c_i)$ for $i \in [N]$ and $\boldsymbol{c} \in C$, with C denoting an $\mathrm{MDS}(N, N-4, 5)$ code. Based on the definitions of ζ and the set $B^m_{\boldsymbol{\sigma}}$ in Construction 52, the ith message block $B_i = \zeta(c_i)$ satisfies $\boldsymbol{\sigma} B_i \boldsymbol{\sigma} \in \mathrm{Irr}(m + 2l)$. Then, by Lemma 51, $\boldsymbol{x} \in \mathrm{Irr}(N(m + l) - l)$ and so $\boldsymbol{x} \in \mathcal{C}_{\boldsymbol{\sigma}}$. Therefore, $\mathcal{C}_{MDS} \subseteq \mathcal{C}_{\boldsymbol{\sigma}}$. Suppose the retrieved word is $\boldsymbol{y}$. By Theorem 50, $\mathcal{D}_{\boldsymbol{\sigma}}(\boldsymbol{y})$ suffers either at most two substitutions or at most four erasures of message blocks. Suppose the message block B_i is substituted by another string $\boldsymbol{v}$ of length m. If $\zeta^{-1}(\boldsymbol{v})$ exists, this translates to a substitution of c_i. If not, we define $\zeta^{-1}(B_i)$ as an arbitrary element of $\mathbb{F}_{2^t}$, again leading to a possible substitution of c_i with another symbol. To decode, we can use the MDS decoder on $\zeta^{-1}(\mathcal{D}_{\boldsymbol{\sigma}}(\boldsymbol{y}))$, which relative to $\boldsymbol{c}$ suffers either $\leqslant 2$ substitutions or $\leqslant 4$ erasures. Given that the minimum Hamming distance of the MDS code is 5, the decoder can successfully recover $\boldsymbol{c}$. □

3.3.3 Extension to edit errors

In this subsection, we extend Theorem 53 to include insertion and deletion errors in addition to substitution errors. We do so by showing an insertion can be viewed as a duplication plus a substitution and a deletion as a substitution plus a deduplication.

The duplication-substitution channel discussed so far can be viewed as shown in Figure 3.5i, where in a pre-decoding step, the root of the retrieved string is found and then passed to the

decoder. Recall that for $\leqslant 3$-TDs, applying a sequence of duplications to a string $\boldsymbol{s}$ and then removing all repeats is equivalent to removing all repeats from $\boldsymbol{s}$. Hence, The process shown in Figure 3.5i is equivalent to the one shown in Figure 3.5ii. The same equivalence holds if we replace the block representing a substitution error with a block representing an edit error. We can now prove the following corollary to Theorem 53.

Corollary 54. *If $m > \mathcal{L}$, then the error-correcting code $\mathcal{C}_{\text{MDS}}$ in Construction 52 can correct any number of $\leqslant$3-TDs and at most one edit error.*

Proof. Suppose that $\boldsymbol{c} \in \mathcal{C}_{\text{MDS}}$ suffers a sequence of errors consisting of duplications, an edit, and more duplications. Then all repeats are removed from the resulting string. This process is equivalent to applying the first set of duplications and the edit and then removing all the repeats. Denote this sequence of operations applied to $\boldsymbol{c}$ by S and the final result by $S(\boldsymbol{c})$. We show that we can find a sequence S' consisting of duplications, at most one *substitution*, and removal of all repeats, such that $S(\boldsymbol{c}) = S'(\boldsymbol{c})$. If the edit in S is a substitution, then we let $S' = S$. If it is an insertion or a deletion, we again start by setting $S' = S$ and then modify S' as follows: i) If the edit is an insertion, we replace it in S' by a duplication and, if needed, a substitution. Namely, we duplicate the symbol before the insertion and then substitute the copy as needed. For example, if in S we have $abc \xrightarrow{ins} abxc$, we replace this step by $abc \xrightarrow{dup} abbc \xrightarrow{sub} abxc$, where *ins* stands for insertion, *dup* for duplication, and *sub* for substitution. The substitution is not necessary if $x = b$. ii) If the edit is a deletion, we replace it by a deduplication that is preceded by a substitution if needed. Namely, we substitute the symbol that is deleted in S to be equal to the previous symbol and then deduplicate it. For example, if in S we have $abc \xrightarrow{del} ac$, we replace this step by $abc \xrightarrow{sub} aac \xrightarrow{dedup} ac$ in S', where *del* stands for deletion and *dedup* for deduplication. If $b = a$, the substitution is not needed. Now, S' is a sequence consisting of duplications, at most one substitution, and then removal of all repeats, and furthermore $S(\boldsymbol{c}) = S'(\boldsymbol{c})$. From Theorem 53 and the above discussion about the equivalence of Figures 3.5i and 3.5i, we find that the decoder can produce $\boldsymbol{c}$ given $S'(\boldsymbol{c})$. □

3.3.4 Construction of message blocks

In this subsection, we study the set $\mathcal{B}^m_{\boldsymbol{\sigma}}$ of valid message blocks of length m with $\boldsymbol{\sigma}$ as the marker. Since in Construction 52, the markers $\boldsymbol{\sigma}$ do not contribute to the size of the code, to maximize the code rate, we set $l = |\boldsymbol{\sigma}| = 5$, i.e., $\boldsymbol{\sigma} \in \text{Irr}(5)$.

For a given $\boldsymbol{\sigma}$, we need to find the set $\mathcal{B}^m_{\boldsymbol{\sigma}}$. The first step in this direction is finding all irreducible sequences of length $m + 2l = m + 10$. We will then identify those that start and end with $\boldsymbol{\sigma}$ but contain no other $\boldsymbol{\sigma}$s.

As shown in [30], the set of $\leqslant$3-irreducible strings over an alphabet of size q is a regular language whose graph $G_q = (V_q, \xi_q)$ is a subgraph of the De Bruijn graph. The vertex set V_q consists of 5-tuples $a_1a_2a_3a_4a_5$ that do not have any repeats (of length at most 2). There is an edge from $a_1a_2a_3a_4a_5 \to a_2a_3a_4a_5a_6$ if $a_1a_2a_3a_4a_5a_6$ belongs to Irr(6). The label for this edge is a_6. The label for a path is the 5-tuple representing its starting vertex concatenated with the labels of the subsequent edges. In this way, the label of a path in this graph is an irreducible sequence and each

irreducible sequence is the label of a unique path in the graph. The graph G_q, when $q = 3$, can be found in [30, Fig. 1].

The following theorem characterizes the set $\mathcal{B}_{\boldsymbol{\sigma}}^m$ and will be used in the next subsection to find the size of the code.

Theorem 55. *Over an alphabet of size q and for $\boldsymbol{\sigma} \in \mathrm{Irr}(5)$, there is a one-to-one correspondence between $B \in \mathcal{B}_{\boldsymbol{\sigma}}^m$ and paths of length $m+5$ in G_q that start and end in $\boldsymbol{\sigma}$ but do not visit $\boldsymbol{\sigma}$ in their interiors. Specifically, each sequence $B \in \mathcal{B}_{\boldsymbol{\sigma}}^m$ corresponds to the path with the label $\boldsymbol{\sigma}B\boldsymbol{\sigma}$.*

Proof. Consider a path $p = \boldsymbol{v}_1\boldsymbol{v}_2\cdots\boldsymbol{v}_{k+1}$ where $\boldsymbol{v}_i$ are vertices of G_q and k is the length of the path. Denote the label of this path by $\boldsymbol{s} = s_1 s_2 \cdots s_{k+5}$. It can be shown by induction on k that $\boldsymbol{v}_i = s_i s_{i+1} s_{i+2} s_{i+3} s_{i+4}$. Hence, the label of a path of length $m+5$ that starts and ends in $\boldsymbol{\sigma}$ but does not visit $\boldsymbol{\sigma}$ otherwise is an irreducible sequence with exactly two occurrences of $\boldsymbol{\sigma}$ and is of the form $\boldsymbol{\sigma}B\boldsymbol{\sigma}$ where $B \in \mathcal{B}_{\boldsymbol{\sigma}}^m$. Conversely, suppose $B \in \mathcal{B}_{\boldsymbol{\sigma}}^m$. Then $\boldsymbol{\sigma}B\boldsymbol{\sigma}$ is an irreducible string of length $m+10$ and thus the label of a unique path of length $m+5$ in G_q. This path starts and ends in $\boldsymbol{\sigma}$. But it does not visit $\boldsymbol{\sigma}$ in its interior since that would imply there are more than two occurrences of $\boldsymbol{\sigma}$ in $\boldsymbol{\sigma}B\boldsymbol{\sigma}$. □

3.3.5 Code rate

We now turn to find the rate of the code introduced in this section. For a code C of length n and size $|C|$, the rate is defined as $R(C) = \frac{1}{n}\log\|C\|$. For the code of Construction 52,

$$R(\mathcal{C}_{\mathrm{MDS}}) = \frac{N-4}{Nm+(N-1)l}\log(N+1),$$

where N depends on the choice of $\boldsymbol{\sigma} \in \mathrm{Irr}(5)$. More specifically, $N \leqslant 2^{\lfloor \log M_{\boldsymbol{\sigma}}^{(m)} \rfloor} - 1$. Choosing the largest permissible value for N implies that $N \geqslant (M_{\boldsymbol{\sigma}}^{(m)} - 1)/2$ and

$$\begin{aligned} R(\mathcal{C}_{\mathrm{MDS}}) &\geqslant \frac{1-4/N}{m+l}\log(N+1) \\ &\geqslant \frac{1}{m+l}\left(1 - \frac{8}{M_{\boldsymbol{\sigma}}^{(m)}-1}\right)(\log M_{\boldsymbol{\sigma}}^{(m)} - 1). \end{aligned}$$

If we let m and $M_{\boldsymbol{\sigma}}^{(m)}$ grow large, the rate becomes

$$R(\mathcal{C}_{\mathrm{MDS}}) = \frac{1}{m}\log M_{\boldsymbol{\sigma}}^{(m)}(1+o(1)). \tag{3.6}$$

For a given alphabet Σ_q, let A denote the adjacency matrix of G_q, where the rows and columns of A are indexed by $\boldsymbol{v} \in V_q \subseteq \Sigma_q^5$. Furthermore, let $A_{(\boldsymbol{v})}$ be obtained by deleting the row and column corresponding to $\boldsymbol{v}$ from A and $c_{(\boldsymbol{v})}$ (resp. $r_{(\boldsymbol{v})}^T$) be the column (row) of A corresponding to $\boldsymbol{v}$ with the element corresponding to $\boldsymbol{v}$ removed. Recall that $M_{\boldsymbol{\sigma}}^{(m)} = |\mathcal{B}_{\boldsymbol{\sigma}}^m|$ and $l = 5$. From Theorem 55, $M_{\boldsymbol{\sigma}}^{(m)}$ equals the number of paths of length $m+l$ in G_q that start and end with $\boldsymbol{\sigma}$ but do not visit $\boldsymbol{\sigma}$ in their interiors. The number of paths of length $m+l-2$ from vertex $\boldsymbol{u}$ to vertex $\boldsymbol{v}$ in G_q without

the occurrence of $\boldsymbol{\sigma}$ is given by the $(\boldsymbol{u}, \boldsymbol{v})$ element of $\left(A_{(\boldsymbol{\sigma})}\right)^{m+l-2}$. Noting that the paths start and end $\boldsymbol{\sigma}$, we have

$$M_{\boldsymbol{\sigma}}^{(m)} = r_{(\boldsymbol{\sigma})}^T \left(A_{(\boldsymbol{\sigma})}\right)^{m+l-2} c_{(\boldsymbol{\sigma})}, \tag{3.7}$$

where $(\cdot)^T$ denotes matrix transpose. Here, multiplying by $r_{(\boldsymbol{\sigma})}^T$ from the left and $c_{(\boldsymbol{\sigma})}$ from the right allow us to sum over elements $(\boldsymbol{u}, \boldsymbol{v})$ of $\left(A_{(\boldsymbol{\sigma})}\right)^{m+l-2}$ such that there is an edge from $\boldsymbol{\sigma}$ to $\boldsymbol{u}$ and an edge from $\boldsymbol{v}$ to $\boldsymbol{\sigma}$. As $m \to \infty$, if $A_{(\boldsymbol{\sigma})}$ is primitive [48], we have

$$\frac{1}{m+l} \log M_{\boldsymbol{\sigma}}^{(m)} \to \log(\lambda_{\boldsymbol{\sigma}}), \tag{3.8}$$

where $\lambda_{\boldsymbol{\sigma}}$ is the largest eigenvalue of $A_{(\boldsymbol{\sigma})}$. Maximizing over $\boldsymbol{\sigma} \in V_q$ yields the largest value for $M_{\boldsymbol{\sigma}}^{(m)}$ in (3.7) and (3.8), and thus the highest code rate. This is possible to do computationally for small values of q and, in particular, for $q = 4$, which corresponds to data storage in DNA. In this case, $A_{(\boldsymbol{\sigma})}$ is primitive for all choices of $\boldsymbol{\sigma} \in \mathrm{Irr}(5)$ and the largest eigenvalue is obtained for $\boldsymbol{\sigma} = 01201$ (and strings obtained from 01201 by relabeling the alphabet symbols). For this $\boldsymbol{\sigma}$, we find $\lambda_{\boldsymbol{\sigma}} = 2.6534$, leading to an asymptotic code rate of 1.4078 bits/symbol.

It was shown in [30] that the set of $\leqslant 3$-irreducible strings of length n is a code correcting any number of $\leqslant$3-TDs. In [35], it was shown that the rate of this code, $\frac{1}{n} \log \| \mathrm{Irr}(n) \|$, is asymptotically optimal. It is easy to see that $\frac{1}{n} \log \| \mathrm{Irr}(n) \| \leqslant \log(q-1)$ as no symbol can be repeated. For the case of $q = 4$, we have $\frac{1}{n} \log \| \mathrm{Irr}(n) \| = \log 2.6590 = 1.4109$ bits/symbol. Therefore, the cost of protection against a single edit in our construction is only 0.003 bits/symbol. It should be noted, however, that here we have assumed m is large, thus ignoring the overhead from the MDS code and marker strings.

In addition to the computational rate obtained above for the important case of $q = 4$, we will provide analytical bounds on the code rate. An important quantity affecting the rate of the code is the number of outgoing edges from each vertex in G_q that do not lead to $\boldsymbol{\sigma}$. The asymptotic rate of the code is bounded from below by the number of such edges. The next lemma, which establishes the number of outgoing edges for each vertex, will be useful in identifying an appropriate choice of $\boldsymbol{\sigma}$, and the following theorem provides a lower bound for $M_{\boldsymbol{\sigma}}^{(m)}$ for such a choice.

Lemma 56. *For $q > 2$, a vertex $\boldsymbol{v} = a_1a_2a_3a_4a_5$ in G_q has $q-2$ outgoing edges if $a_3 = a_5$ or $a_1a_2 = a_4a_5$. Otherwise, it has $q-1$ outgoing edges.*

Proof. Consider $\boldsymbol{v} = a_1a_2a_3a_4a_5 \in \mathrm{Irr}(5)$, and $\boldsymbol{w} = a_2a_3a_4a_5a_6 \in \mathrm{Irr}(5)$. There is an edge from $\boldsymbol{v}$ to $\boldsymbol{w}$ if $a_1a_2a_3a_4a_5a_6 \in \mathrm{Irr}(6)$. The number of outgoing edges from $\boldsymbol{v}$ equals the number of possible values for a_6 such that this condition is satisfied. Clearly, $a_6 \neq a_5$. Furthermore, if $a_3 = a_5$, then $a_6 \neq a_4$ and if $a_1a_2 = a_4a_5$, then $a_6 \neq a_3$.

However, $a_3 = a_5$ and $a_1a_2 = a_4a_5$ cannot simultaneously hold, since that would imply $a_2 = a_3$, contradicting $\boldsymbol{v} \in \mathrm{Irr}(5)$. Hence, if either $a_3 = a_5$ or $a_1a_2 = a_4a_5$ holds, then there are $q-2$ outgoing edges and if neither holds, there are $q-1$ outgoing edges. □

Since $\boldsymbol{\sigma}$ must also be excluded, it may seem that the number of outgoing edges may be as low as $q-3$. But we show in the next theorem that with an appropriate choice of $\boldsymbol{\sigma}$, we can have $q-2$

as the lower bound.

Theorem 57. *Over an alphabet of size $q > 2$, there exists $\boldsymbol{\sigma} \in \mathrm{Irr}(5)$ such that $M_{\boldsymbol{\sigma}}^{(m)} \geqslant (q-2)^{m-c_q}$, where c_q is a constant independent from m.*

Proof. Recall that $M_{\boldsymbol{\sigma}}^{(m)}$ is the number of paths of length $m+5$ in G_q that start and end in $\boldsymbol{\sigma}$ but do not visit $\boldsymbol{\sigma}$ otherwise. Since the path must return to $\boldsymbol{\sigma}$, we will show below that for an appropriate choice of $\boldsymbol{\sigma}$, there is a path in G_q from any vertex to $\boldsymbol{\sigma}$, and define c_q such that the length of this path is at most $c_q + 5$. Hence $M_{\boldsymbol{\sigma}}^{(m)}$ is at least the number of paths of length $m - c_q$ from $\boldsymbol{\sigma}$ to another vertex that do not pass through $\boldsymbol{\sigma}$.

As shown in Lemma 56, each vertex in G_q has at least $q-2$ outgoing edges. We select $\boldsymbol{\sigma}$ such that this still holds even if edges leading to $\boldsymbol{\sigma}$ are excluded. We do so by ensuring that each vertex $\boldsymbol{v}$ with an outgoing edge to $\boldsymbol{\sigma}$ has $q-1$ outgoing edges. Let $\boldsymbol{v} = a_1a_2a_3a_4a_5$ and $\boldsymbol{\sigma} = a_2a_3a_4a_5a_6$. Based on Lemma 56, if $a_2 \neq a_5$ and $a_3 \neq a_5$, then $\boldsymbol{v}$ has $q-1$ outgoing edges. In particular, we can choose $\boldsymbol{\sigma} = 01020$ since $q \geqslant 3$. With this choice, $M_{\boldsymbol{\sigma}}^{(m)} \geqslant (q-2)^{m-c_q}$.

To complete the proof, we need to show that there is a path in G_q from any vertex to $\boldsymbol{\sigma} = 01020$. For $q = 3, 4, 5$, we have checked this claim computationally by explicitly forming G_q. Let us then suppose $q \geqslant 6$, where the alphabet Σ_q contains $\{3, 4, 5\}$. Let $\boldsymbol{v} = a_1 \cdots a_5$ be some vertex in G_q. There is an edge from $\boldsymbol{v}$ to $a_2 \cdots a_6$ for some $a_6 \in \{3, 4, 5\}$ since, from Lemma 56, at most two elements of Σ_q are not permissible. Continuing in similar fashion, in 5 steps, we can go from $\boldsymbol{v}$ to some vertex $\boldsymbol{w} = b_1 \cdots b_5$ whose elements b_i belong to $\{3, 4, 5\}$. We can then reach $\boldsymbol{\sigma}$ in 5 additional steps via the path $\boldsymbol{w} \to b_2 \cdots b_4b_5 0 \to b_3b_4b_501 \to \cdots \to \boldsymbol{\sigma}$, proving the claim. In particular, for $q \geqslant 6$, we have $c_q \leqslant 5$. □

We can now find a lower bound on the asymptotic rate, based on (3.6) and the proceeding theorem:

Corollary 58. *For $q > 2$, as $m \to \infty$, $R(\mathcal{C}_{\mathrm{MDS}}) \geqslant \log(q-2)(1 + o(1))$.*

We note that this gives the lower bound of 1 bit/symbol for $q = 4$, which we can compare to the upper bound of $\log(q-1) = 1.585$ for codes correcting only duplications and to the rate obtained computationally following (3.8), which was 1.4078 bits/symbol.

3.4 Correcting short duplications and at most p substitutions

Based on the work in Section 3.3, one interesting question is to correct more than one edit or substitution error apart from many short duplications. In this subsection, we construct error-correcting codes to correct an arbitrary number of $\leqslant$3-TDs and at most p substitutions. The key idea is to concatenate every $3p$ consecutive message blocks with different "colors" in Construction 48 as one *message group* such that at most $2p$ message groups are substituted or erased in the roots of the output. This section starts by the analysis of the channel properties, followed by the code construction, code rate, and the time complexity.

3.4.1 The channel with short duplications and at most p substitutions

In this subsection, we study channels with an arbitrary number of $\leqslant$3-TDs and at most p substitutions and motivate our error-correction approach.

First, we consider channels with only short duplication errors (i.e., $\leqslant$3-TDs). Let $\boldsymbol{x}$ and $\boldsymbol{y} \in D^*(\boldsymbol{x})$ denote the input and output of the channel. Note that the duplication root of $\boldsymbol{x}$ is also the duplication root of $\boldsymbol{y}$. This fact, along with the uniqueness of duplication roots for short duplications, implies that the channel does not alter the duplication root. This observation was used in [30] to propose using the set of irreducible strings of length n as a code for correcting an arbitrary number of duplications. This code was shown to be asymptotically optimal by [35].

The problem of correcting short duplications and an additional substitution was studied in [84], [85]. There, motivated by the use of roots for correcting duplication errors, the effect of the substitution on the duplication roots was studied.

Theorem 59. *[85, Theorem 3] There exists a (minimal) positive integer $\mathcal{L}$ such that for any $\boldsymbol{x}$ and $\boldsymbol{y} \in D^{*,\leqslant 1}(\boldsymbol{x})$, $R(\boldsymbol{y})$ can be obtained from $R(\boldsymbol{x})$ through an $\mathcal{L}$-substring edit.*

It was shown in [85] that $\mathcal{L} \leqslant 17$. Note that a priori, it is not clear that the effect of the substitution can be limited to a short substring since the substituted symbol may be duplicated as parts of substrings of different lengths an arbitrary number of times.

For the channel with many short duplications and at most p substitutions, we can prove the following result.

Theorem 60. *For any $\boldsymbol{x} \in \Sigma^*$ and $\boldsymbol{y} \in D^{*,\leqslant p}(\boldsymbol{x})$, $R(\boldsymbol{y})$ can be obtained from $R(\boldsymbol{x})$ by at most p $\mathcal{L}$-substring edit errors.*

Proof. Consider the sequence of substitution and duplication errors that transform $\boldsymbol{x}$ into $\boldsymbol{y}$. (Note that the errors may occur in any order.) For $i \in [p]$, let $\boldsymbol{y}_i$ be the sequence obtained just after the ith substitution error. Furthermore, let $\boldsymbol{y}_0 = \boldsymbol{x}$ and $\boldsymbol{y}_{p+1} = \boldsymbol{y}$. By Theorem 59, for $i \in [p]$, $R(\boldsymbol{y}_i)$ can be obtained from $R(\boldsymbol{y}_{i-1})$ via an $\mathcal{L}$-substring edit. Also, $R(\boldsymbol{y}_p) = R(\boldsymbol{y}_{p+1})$. Hence, $R(\boldsymbol{y}_0)$ can be transformed into $R(\boldsymbol{y}_{p+1})$ via p $\mathcal{L}$-substring edits. □

We next provide an example, demonstrating Theorem 60.

Example 61. *Let the alphabet be $\Sigma_4 = \{0, 1, 2, 3\}$ and $p = 2$. We take the input $\boldsymbol{x}$ to be irreducible, i.e., $R(\boldsymbol{x}) = \boldsymbol{x}$. By passing through the channel, $\boldsymbol{x}$ suffers multiple $\leqslant$3-TDs and 2 symbol substitutions, resulting in $\boldsymbol{y} \in D^{*,2}(\boldsymbol{x})$. We show the difference between $R(\boldsymbol{x})$ and $R(\boldsymbol{y})$ for two possible input-output pairs. Below, substrings added via duplication are marked with underlines, while substituted symbols are red and bold.*

First, we provide an example where $R(\boldsymbol{y})$ can be obtained from $R(\boldsymbol{x})$ via non-overlapping substring edits:

$$\boldsymbol{x} = 3210313230121321,$$
$$\boldsymbol{y} = 3213\underline{20}3210\underline{3}1\underline{3}132\underline{13}23\mathbf{2}12132\underline{13}21,$$

$$R(\boldsymbol{x}) = \underbrace{321}_{\boldsymbol{\alpha}_0}\underbrace{}_{\boldsymbol{\beta}_1}\underbrace{031}_{\boldsymbol{\alpha}_1}\underbrace{3230121}_{\boldsymbol{\beta}_2}\underbrace{321}_{\boldsymbol{\alpha}_2},$$
$$R(\boldsymbol{y}) = \underbrace{321}_{\boldsymbol{\alpha}_0}\underbrace{32\mathbf{0}321}_{\boldsymbol{\beta}'_1}\underbrace{031}_{\boldsymbol{\alpha}_1}\underbrace{}_{\boldsymbol{\beta}'_2}\underbrace{321}_{\boldsymbol{\alpha}_2},$$

where the errors are $\boldsymbol{\beta}_1 = \Lambda \to \boldsymbol{\beta}'_1$ *and* $\boldsymbol{\beta}_2 \to \boldsymbol{\beta}'_2 = \Lambda$.

In the second case, the two edits overlap, leading to a single substring substitution:

$$\boldsymbol{x} = 132031230,$$
$$\boldsymbol{y} = 132\underline{32}03\mathbf{2}1\underline{32}\mathbf{0}\underline{321}230\underline{230230},$$
$$R(\boldsymbol{x}) = \underbrace{13203}_{\boldsymbol{\alpha}_0}\underbrace{}_{\boldsymbol{\beta}_1}\underbrace{1230}_{\boldsymbol{\alpha}_1}$$
$$R(\boldsymbol{y}) = \underbrace{13203}_{\boldsymbol{\alpha}_0}\underbrace{\mathbf{2}132\mathbf{0}32}_{\boldsymbol{\beta}'_1}\underbrace{1230}_{\boldsymbol{\alpha}_1}.$$

Generally, t overlapping $\mathcal{L}$-substring edits result in a $(t\mathcal{L})$-substring edit.

3.4.2 Code construction

To construct codes correcting at most p $\mathcal{L}$-substring edits in irreducible sequences, similar to [83], we divide the codewords into message blocks, separated by markers, while maintaining irreducibility, such that an $\mathcal{L}$-substring edit only affects a limited number of message blocks. In the case of $p = 1$ studied in [83], it was shown that if the markers appear in the correct positions in the retrieved word, then at most two of the message blocks are substituted. For $p > 1$ however, even if all markers are in the correct positions, it is not guaranteed that a limited number of message blocks are substituted, making it challenging to correct more than one error.

We start by recalling an auxiliary construction from [83].

Construction 62. *[83, Construction 6] Let l, m, N_B be positive integers with $m > l \geqslant 5$ and $\boldsymbol{\sigma} \in \mathrm{Irr}_q(l)$. Also, let $\mathcal{B}^m_{\boldsymbol{\sigma}}$ denote the set of sequences B of length m such that $\boldsymbol{\sigma}B\boldsymbol{\sigma}$ is irreducible and has exactly two occurrences of $\boldsymbol{\sigma}$. Define*

$$\mathcal{C}_{\boldsymbol{\sigma}} = \{B_1\boldsymbol{\sigma}B_2\boldsymbol{\sigma}\cdots\boldsymbol{\sigma}B_{N_B} : B_i \in \mathcal{B}^m_{\boldsymbol{\sigma}}\}.$$

The irreducibility of $\boldsymbol{\sigma}B_i\boldsymbol{\sigma}$ ensures that the codewords are irreducible.

We denote the output of the channel by $\boldsymbol{y}$. Define a *block* in $\boldsymbol{y}$ as a maximal substring that does not overlap with any $\boldsymbol{\sigma}$. Furthermore, define an *m-block* in $\boldsymbol{y}$ as a block of length m. Note that m-blocks can be either message blocks in $\boldsymbol{x}$ or new blocks created by substring edits.

Having divided each codeword into N_B message blocks and $N_B - 1$ separators, we study in the next lemma how message blocks are affected by the errors.

Lemma 63. *Let $\boldsymbol{x} \in \mathcal{C}_{\boldsymbol{\sigma}}$, $m > \mathcal{L}$, and $\boldsymbol{y}$ be generated from $\boldsymbol{x}$ through at most p $\mathcal{L}$-substring edits. Then there are less than $(N_B + p)$ m-blocks in $\boldsymbol{y}$. Furthermore, there are at least $N_B - 2p$ error-free*

m-blocks in $\boldsymbol{y}$ which appear in $\boldsymbol{x}$ in the same order. More precisely, there are blocks $B_{i_1}, B_{i_2}, \ldots, B_{i_k}$ in $\boldsymbol{y}$, where $k \geqslant N_B - 2p$, each B_{i_j} is a message block in $\boldsymbol{x}$, and any two blocks B_{i_j} and $B_{i_{j'}}$ have the same relative order of appearance in $\boldsymbol{x}$ and in $\boldsymbol{y}$.

Proof. First suppose $\boldsymbol{y}$ has $\geqslant (N_B + p)$ message blocks. This implies that the length of $\boldsymbol{y}$ is at least $(N_B + p)m + (N_B + p - 1)l$, which is larger than the length of $\boldsymbol{x}$ by $pm + (p-1)l$. But this is not possible as $m > \mathcal{L}$ and the total length of inserted substrings is at most $p\mathcal{L}$.

Furthermore, if $m > \mathcal{L}$, each $\mathcal{L}$-substring edit alters i) a message block in $\boldsymbol{x}$, ii) a message block and a marker $\boldsymbol{\sigma}$, or iii) two message blocks and the marker between them. Hence at least $N_B - 2p$ message blocks of $\boldsymbol{x}$ appear in $\boldsymbol{y}$ without being changed. □

If the positions of the error-free m-blocks described in Lemma 63 in $\boldsymbol{y}$ were known, a Reed-Solomon (RS) code of length N_B and dimension $N_B - 2p$ could be used to recover codewords in $\mathcal{C}_{\boldsymbol{\sigma}}$. This however is not the case since the blocks can be shifted by substring edits. In order to determine the positions of the error-free m-blocks, we introduce another auxiliary construction based on Construction 62 by combining message blocks into *message groups*, where the message blocks in each group have different "colors".

Construction 64. *For an integer T, we partition $\mathcal{B}^m_{\boldsymbol{\sigma}}$ into T parts $\mathcal{B}^m_{\boldsymbol{\sigma}}(j), j \in [T]$. The elements of $\mathcal{B}^m_{\boldsymbol{\sigma}}(j)$ are said to have* color *j. Let N_B be a positive integer that is divisible by T. We define the code*

$$\mathcal{C}_{(\boldsymbol{\sigma},T)} = \left\{ B_1\boldsymbol{\sigma}B_2\boldsymbol{\sigma}\cdots\boldsymbol{\sigma}B_{N_B} \in \mathcal{C}_{\boldsymbol{\sigma}} : B_i \in \mathcal{B}^m_{\boldsymbol{\sigma}}(i \mod' T) \right\},$$

where $\mathcal{C}_{\boldsymbol{\sigma}}$ has parameters m, l with $m > \mathcal{L}$ and $m > l \geqslant 5$.

We divide the message blocks $B_1, \ldots, B_{N_B}$ in each $\boldsymbol{x} \in \mathcal{C}_{(\boldsymbol{\sigma},T)}$ into $\hat{N} = N_B/T$ message groups*, where the k-th message group is $S_k = (B_{(k-1)T+1}, \ldots, B_{kT-1}, B_{kT})$. Note that the message blocks in each message group have colors $1, 2, \ldots, T$ in order.*

For example, if $N_B = 12, T = 3, \hat{N} = 4$, then in a codeword

$$\boldsymbol{x} = B_1\boldsymbol{\sigma}B_2\boldsymbol{\sigma}B_3\boldsymbol{\sigma}B_4\boldsymbol{\sigma}B_5\boldsymbol{\sigma}B_6\boldsymbol{\sigma}\cdots\boldsymbol{\sigma}B_{10}\boldsymbol{\sigma}B_{11}\boldsymbol{\sigma}B_{12},$$

the first group is (B_1, B_2, B_3) and the second group is (B_4, B_5, B_6). Furthermore, message blocks in both groups have colors $(1, 2, 3)$. The colors in the message group will help us identify the true positions of the message blocks.

Definition 65. *For $\boldsymbol{x} \in \mathcal{C}_{(\boldsymbol{\sigma},T)}$ and $\boldsymbol{y}$ derived from $\boldsymbol{x}$ through at most p $\mathcal{L}$-substring edits, let the i-th m-block in $\boldsymbol{y}$ be denoted by B'_i. A* T-group *in $\boldsymbol{y}$ is a substring $B'_{k+1}\boldsymbol{\sigma}B'_{k+2}\cdots\boldsymbol{\sigma}B'_{k+T}$ such that the m-block B'_{k+j} has color j.*

The next lemma characterizes how error-free message groups (those that do not suffer any substring edits but may be shifted) appear in $\boldsymbol{y}$.

Lemma 66. *Suppose $\boldsymbol{x} \in \mathcal{C}_{(\boldsymbol{\sigma},T)}$ and let $\boldsymbol{y}$ be obtained from $\boldsymbol{x}$ through at most p $\mathcal{L}$-substring edits. For $r \in [\hat{N}]$, if the r-th message group in $\boldsymbol{x}$ is not affected by any substring edit errors, then it will appear as a T-group after b m-blocks in $\boldsymbol{y}$, where $b \in [(r-1)T - 2p, (r-1)T + p - 1]$.*

Proof. Since $m > \mathcal{L}$, each $\mathcal{L}$-substring edit can affect at most two message blocks and thus at most two message groups. Hence, there are at least $\hat{N} - 2p$ message groups that do not suffer any substring edits.

Let the r-th message group S_r in $\boldsymbol{x}$ be free of substring edits. Given that the colors of its message blocks are not altered, it will appear as a T-group in $\boldsymbol{y}$. Since each substring edit alters at most two message blocks, among the $(r-1)T$ message blocks appearing before S_r in $\boldsymbol{x}$, at most $2p$ do not appear in $\boldsymbol{y}$. Furthermore, the substring edits add at most $p\mathcal{L}$ to the length of $\boldsymbol{x}$. Since $m > \mathcal{L}$, this means that at most $p-1$ new m-blocks are created in $\boldsymbol{y}$. Hence, $b \in [(r-1)T-2p, (r-1)T+p-1]$. □

The previous lemma guarantees the presence of error-free, but possibly shifted, T-groups, and provides bounds on their position in $\boldsymbol{y}$. In the following theorem, we use these facts to show that these T-groups can be synchronized and the errors can be localized.

Theorem 67. *Let $\mathcal{C}_{(\boldsymbol{\sigma},T)}$ be a code in Construction 64 and suppose $T \geqslant 3p$ and $\hat{N} \geqslant 4p+1$. There is a decoder* Dec *such that, for any $\boldsymbol{x} \in \mathcal{C}_{(\boldsymbol{\sigma},T)}$ and $\boldsymbol{y}$ derived from $\boldsymbol{x}$ through at most p $\mathcal{L}$-substring edits, $\boldsymbol{v} = \mathrm{Dec}(\boldsymbol{y})$ suffers at most t substitutions and e erasures of message groups, where $t+e \leqslant 2p$.*

Proof. We start by identifying all T-groups in $\boldsymbol{y}$. Note that no two T-groups can overlap. Let $\boldsymbol{v} = (S'_1, \ldots, S'_{\hat{N}})$ be the decoded vector, where S'_r is the decoded version of the message group S_r, determined as follows.

For $r = 1, \ldots, \hat{N}$:

1. If there exists a T-group $\mathcal{T}$ appearing after b message blocks such that $b \in [(r-1)T - 2p, (r-1)T + p - 1]$, then let $S'_r = \mathcal{T}$.

2. If such a T-group does not exist, let $S'_r = \Lambda$, denoting an erasure.

We note that for each r, at most one T-group may satisfy the condition in 1). If two such T-groups exist appearing after b and b' message blocks, we must have $|b - b'| \geqslant T$ and $b, b' \in [(r-1)T - 2p, (r-1)T + p - 1]$, implying $3p - 1 \geqslant T$, which contradicts the assumption on T.

If a message group S_r is not subject to a substring edit, then by Lemma 66, we have $S'_r = S_r$. Otherwise, we may have a substitution of that message group, i.e., $S'_r \neq S_r$, or an erasure, $S'_r = \Lambda$. Since each substring edit may affect at most 2 message groups, the total number of substitutions and erasures is no more than $2p$. □

We now construct an MDS code that can correct the output of the decoder of Theorem 67.

Construction 68. *Let $\mathcal{C}_{(\boldsymbol{\sigma},T)}$ be the code in Construction 64 with parameters $l, m, T, \hat{N}$ satisfying $m > \mathcal{L}, m > l \geqslant 5, T \geqslant 3p$, and $\hat{N} \geqslant 4p+1$. Furthermore, assume $|\mathcal{B}^m_{\boldsymbol{\sigma}}(j)| \geqslant \hat{N}+1$ for $j \in [T]$. Finally, let γ be a positive integer such that $2^\gamma \leqslant \hat{N}+1$ and $\zeta_j : \mathbb{F}_{2^\gamma} \to \mathcal{B}^m_{\boldsymbol{\sigma}}(j)$ be an injective*

mapping for $j \in [T]$. We define $\mathcal{C}_E$ as

$$\begin{aligned}\mathcal{C}_E = \{&\zeta_1(c_1^1)\boldsymbol{\sigma}\cdots\boldsymbol{\sigma}\zeta_j(c_1^j)\boldsymbol{\sigma}\cdots\boldsymbol{\sigma}\zeta_T(c_1^T)\boldsymbol{\sigma}\\ &\zeta_1(c_2^1)\boldsymbol{\sigma}\cdots\boldsymbol{\sigma}\zeta_j(c_2^j)\boldsymbol{\sigma}\cdots\boldsymbol{\sigma}\zeta_T(c_2^T)\boldsymbol{\sigma}\cdots\\ &\zeta_1(c_{\hat{N}}^1)\boldsymbol{\sigma}\cdots\boldsymbol{\sigma}\zeta_j(c_{\hat{N}}^j)\boldsymbol{\sigma}\cdots\boldsymbol{\sigma}\zeta_T(c_{\hat{N}}^T):\\ &\{\boldsymbol{c}^j, j\in[T]\}\subseteq \mathrm{MDS}(\hat{N},\hat{N}-4p,4p+1)\},\end{aligned}$$

where $\mathrm{MDS}(\hat{N},\hat{N}-4p,4p+1)$ denotes an MDS code over $\mathbb{F}_{2^\gamma}$ of length $\hat{N}=2^\gamma-1$, dimension $\hat{N}-4p$, and minimum Hamming distance $d_H=4p+1$, and $\boldsymbol{c}^j=(c_1^j,c_2^j,\ldots,c_{\hat{N}}^j)$ is a codeword of the MDS code.

For each j, we also define an inverse ζ_j^{-1} for ζ_j. For $B\in\mathcal{B}_{\boldsymbol{\sigma}}^m(j)$, if $\beta\in\mathbb{F}_{2^\gamma}$ such that $\zeta_j(\beta)=B$ exists, then let $\zeta_j^{-1}(B)=\beta$. Otherwise, let $\zeta_j^{-1}(B)=0$.

Theorem 69. *The error-correcting codes $\mathcal{C}_E$ in Construction 68 can correct any number of short duplications and at most p symbol substitutions.*

Proof. Given a codeword $\boldsymbol{x}\in\mathcal{C}_E$, let $\boldsymbol{x}''\in\mathcal{D}^{\leqslant p}(\boldsymbol{x})$ and let $\boldsymbol{y}=R(\boldsymbol{x}'')$. Note that by construction, $\boldsymbol{x}$ is irreducible. Thus, by Theorem 82, $\boldsymbol{y}$ can be obtained from $\boldsymbol{x}$ through at most p $\mathcal{L}$-substring edits. As $\mathcal{C}_E\subseteq\mathcal{C}_{(\boldsymbol{\sigma},T)}$, based on Theorem 67, $\boldsymbol{v}=\mathrm{Dec}(\boldsymbol{y})$ suffers at most t substitutions and e erasures of message groups, where $t+e\leqslant 2p$. Hence, for $j\in[T]$, the blocks $(\zeta_j(c_1^j),\zeta_j(c_2^j),\ldots,\zeta_j(c_{\hat{N}}^j))$ suffer at most $2p$ erasures or substitutions. Consequently, if we apply ζ_j^{-1} to the corresponding retrieved blocks in $\boldsymbol{v}$, the codeword $(c_1^j,c_2^j,\ldots,c_{\hat{N}}^j)$ also suffers at most $2p$ substitutions or erasures, which can be corrected using the MDS code. □

3.4.3 Code rate

In this subsection, we present choices for the parameters of Construction 68 and discuss the rate of the resulting code.

Among the n_E symbols of each codeword in Construction 68, $4pTm+(\hat{N}T-1)l$ symbols belong to MDS parities or markers. We choose T and l to be their smallest possible values and set $T=3p$ and $l=5$.

The construction requires that $\|\mathcal{B}_{\boldsymbol{\sigma}}^m(j)\|\geqslant\hat{N}+1$ for all j. Let $M_{\boldsymbol{\sigma}}^{(m)}=\|\mathcal{B}_{\boldsymbol{\sigma}}^m\|$. Dividing $\mathcal{B}_{\boldsymbol{\sigma}}^m$ into parts of nearly equal sizes, we find that each part $\mathcal{B}_{\boldsymbol{\sigma}}^m(j)$ has size at least $M_{\boldsymbol{\sigma}}^{(m)}/T-1$. We then choose $\hat{N}+1$ as the largest power of two not larger than $M_{\boldsymbol{\sigma}}^{(m)}/T-1$, ensuring that $\hat{N}+1\geqslant M_{\boldsymbol{\sigma}}^{(m)}/(2T)-(1/2)$. Assume

$$M_{\boldsymbol{\sigma}}^{(m)}\geqslant 24p^2+15p. \tag{3.9}$$

Then $\hat{N}+1\geqslant M_{\boldsymbol{\sigma}}^{(m)}/(2T)-(1/2)\geqslant 4p+2$.

Furthermore, note that $\hat{N}T(m+5)-5=n_E$ and thus $\hat{N}=\frac{n_E+5}{(m+5)(3p)}$. The size of the code then becomes

$$\|\mathcal{C}_E\|=(\hat{N}+1)^{(\hat{N}-4p)(3p)},$$

and

$$\begin{aligned}\log \|\mathcal{C}_E\| &\geqslant \left(\frac{n_E}{m+5} - 12p^2\right)\log\left(\frac{M_{\boldsymbol{\sigma}}^{(m)}}{6p} - \frac{1}{2}\right)\\ &\geqslant \left(\frac{n_E}{m+5} - 12p^2\right)\left(\log M_{\boldsymbol{\sigma}}^{(m)} + \log\left(\frac{1}{6p} - \frac{1}{2M_{\boldsymbol{\sigma}}^{(m)}}\right)\right)\\ &\geqslant \left(\frac{n_E}{m+5} - 12p^2\right)\Big(\log M_{\boldsymbol{\sigma}}^{(m)} - \log(6p+1)\Big), \end{aligned} \tag{3.10}$$

where in the last step we have used the fact that $M_{\boldsymbol{\sigma}}^{(m)} \geqslant 24p^2 + 15p$.

It was shown in [83] that $M_{\boldsymbol{\sigma}}^{(m)} \geqslant (q-2)^{m-c_q}$ for some $\boldsymbol{\sigma}$, where c_q is a constant independent of m. In particular, $c_3 \leqslant 13, c_4 \leqslant 7, c_5 \leqslant 6$, and $c_q \leqslant 5$ for $q \geqslant 6$. To satisfy (3.9), we need

$$m \geqslant \max\{\log_{q-2}(24p^2 + 15p) + c_q, \mathcal{L} + 1\}. \tag{3.11}$$

From (3.10), for the rate of $\mathcal{C}_E$,

$$\begin{aligned}\frac{\log \|\mathcal{C}_E\|}{n_E} &\geqslant \left(\frac{m - c_q}{m+5} - \frac{12p^2 m}{n_E}\right)\log(q-2) - \frac{\log(6p+1)}{m+5}\\ &\geqslant \left(1 - \frac{c_q + 5}{m+5} - \frac{12p^2 m}{n_E}\right)\log(q-2) - \frac{\log(6p+1)}{m+5},\end{aligned}$$

where m satisfies (3.11). For $\log p = o(\log n_E)$, letting $m = \Theta(\log n_E)$, we find that the rate asymptotically satisfies

$$\frac{\log \|\mathcal{C}_E\|}{n_E} \geqslant \log(q-2)(1 - o(1)),$$

while the redundancy is at least $\Theta(n_E/\log n_E)$. We observe that a low redundancy and an asymptotic rate equal to that of $\mathrm{Irr}_q(n_E)$ is not guaranteed for $\mathcal{C}_E$, unlike $\mathcal{C}^B$, proposed in Construction 90 in the next section. However, $\mathcal{C}^B$ relies on $\mathcal{C}_E$ to protect its syndrome as stated in Lemma 88, whose proof is given in the subsection 3.5.4.

3.4.4 Time complexity of encoding and decoding

In this subsection, we analyze the time complexities of both the encoding and decoding algorithms for the error-correcting code in Construction 68. Recall that we choose T to be a constant and choose $\hat{N} = \Theta(\|\mathcal{B}_{\boldsymbol{\sigma}}^m\|)$ thus satisfying $\log \hat{N} = \Theta(m)$. Also, note that $n_E = \Theta(\hat{N})$. Furthermore, we choose each part $\mathcal{B}_{\boldsymbol{\sigma}}^m(j)$ in the partition of $\mathcal{B}_{\boldsymbol{\sigma}}^m$ to be a contiguous block in the lexicographically sorted list of the elements of $\mathcal{B}_{\boldsymbol{\sigma}}^m$. So the complexity of computing the mapping ζ_j is polynomial in $\|\mathcal{B}_{\boldsymbol{\sigma}}^m\|$ and thus in $\hat{N}$.

We first discuss the complexity of the encoding. The complexity of producing the MDS codewords used in $\mathcal{C}_E$ is polynomial in $\hat{N}$. Mapping these to sequences in $\mathcal{B}_{\boldsymbol{\sigma}}^m$ is also polynomial in $\hat{N}$ as discussed in the previous paragraph. Hence, the encoding complexity is polynomial in $\hat{N}$ as well as in n_E.

Decoding can be performed as described in the proof of Theorem 69, using the decoder described in Theorem 67 and its proof. As the steps described in the proofs of these theorems are polynomial

in the length of the received sequence, so is the time complexity of the decoding.

3.5 Low-redundancy codes to correcting short duplications and at most p edits

Compared to the codes in [30] for only duplications, the codes in Construction 68 incur an asymptotic rate loss when $q = 4$ in order to correct the additional edits. By applying the syndrome compression technique [72]–[75], the current chapter provides codes for correcting any number of short duplications, $\leqslant$3-TDs, and at most p (unrestricted) edits with no asymptotic rate penalty compared to correcting short duplications only, where p and the alphabet size q are constants.

One of the challenging aspects of correcting multiple types of errors, even when optimal codes for individual error types exist, is that codes for each type may utilize incompatible strategies. In particular, correcting duplications relies on constrained codes (local constraints) while edits are corrected using error-correcting codes with codewords that satisfy certain global constraints. Combining these strategies is not straightforward as encoding one set of constraints may violate the other, or alter how errors affect the data. Our strategy, which can be viewed as modified concatenation described in [50], is to first encode user data as a constrained sequence $\boldsymbol{x}$, which does not contain any repeats of length $\leqslant 6$ (such sequences are called irreducible). Then using *syndrome compression*, we compute and append to $\boldsymbol{x}$ a "parity" sequence $\boldsymbol{r}$ to help correct errors that occur in $\boldsymbol{x}$. Syndrome compression has recently been used to provide explicit constructions for correcting a wide variety of errors with redundancy as low as roughly twice the Gilbert-Varshamov bound [72]–[75].

Another challenge arises from the interaction between the errors. When both short duplications and edits are present, a single edited symbol may be duplicated many times and affect an unbounded segment. However, when the input is an irreducible sequence, after removing all tandem copies with length $\leqslant 3$ from the output, the effects of short duplications and at most p edits can be localized in at most p substrings, each with length $\leqslant 17$ [83]. Using the structure of these localized alterations, we describe the set of strings that can be confused with $\boldsymbol{x}$ and bound its size, allowing us to leverage syndrome compression to reduce redundancy.

A third challenge is ensuring that the appended vector $\boldsymbol{r}$ is itself protected against errors and can be decoded correctly. We do this by introducing a higher-redundancy MDS-based code over irreducible sequences. After decoding the appended vector, we use it to recover the data by eliminating incorrect confusable inputs.

Compared to the explicit code for short duplications only [30], the proposed code corrects $\leqslant p$ edits in addition to the duplications at the extra cost of roughly $8p(\log_q n)(1 + o(1))$ symbols of redundancy for $q \geqslant 4$, and achieves the same asymptotic code rate. We note that the state-of-the-art redundancy for correcting p edits is no less than $4p \log_q n(1 + o(1))$ [73]. Time complexities of both the encoding and decoding processes are polynomial when p is a constant.

For simplicity of exposition, we first consider the channel with short duplications and *substitutions* only and construct codes for it. Then, in Subsection 3.5.5, we show that the same codes can

correct short duplications and *edit* errors. We note that short duplications and edits may occur in any order. Henceforth, the term duplication refers to short duplications only.

3.5.1 Notation and preliminaries

Given a sequence $\boldsymbol{x} \in \Sigma_q^n$, we define the binary matrix $\mathcal{U}(\boldsymbol{x})$ of $\boldsymbol{x}$ as

$$\begin{bmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,n} \\ u_{2,1} & u_{2,2} & \cdots & u_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ u_{\lceil \log q \rceil,1} & u_{\lceil \log q \rceil,2} & \cdots & u_{\lceil \log q \rceil,n} \end{bmatrix} \in \{0,1\}^{\lceil \log q \rceil \times n},$$

where the jth column of $\mathcal{U}(\boldsymbol{x})$ is the binary representation of the jth symbol of $\boldsymbol{x}$ for $j \in [n]$, and the ith row of $\mathcal{U}(\boldsymbol{x})$ is denoted as $\mathcal{U}_i(\boldsymbol{x})$ for $i \in \lceil \log q \rceil$.

Suppose $q \geqslant 3$ is a constant. We start with the definition of confusable sets for a given channel and a given set of strings $S \subseteq \Sigma_q^n$. In our application, S is the set of irreducible strings, upon which the proposed codes will be constructed.

Definition 70. *A* confusable set $B(\boldsymbol{x}) \subseteq S$ *of* $\boldsymbol{x} \in S$ *consists of all* $\boldsymbol{y} \in S$*, excluding* $\boldsymbol{x}$*, such that* $\boldsymbol{x}$ *and* $\boldsymbol{y}$ *can produce the same output when passed through the channel.*

Definition 71. *Let* $\mathcal{R}(n)$ *be an integer function of* n*. A* labeling function for the confusable sets $B(\boldsymbol{x}), \boldsymbol{x} \in S$*, is a function*

$$f : \Sigma_q^n \to \Sigma_{2^{\mathcal{R}(n)}}$$

such that, for any $\boldsymbol{x} \in S$ *and* $\boldsymbol{y} \in B(\boldsymbol{x})$*,* $f(\boldsymbol{x}) \neq f(\boldsymbol{y})$*.*

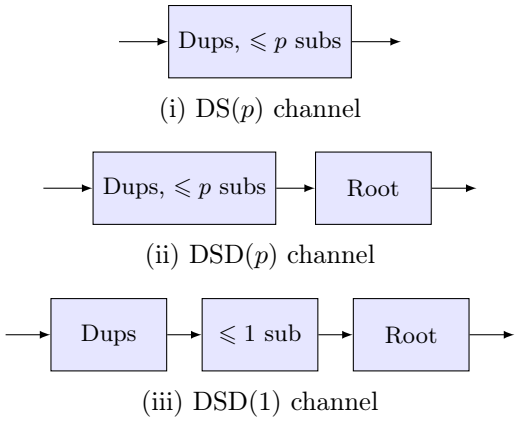
Theorem 72. *(c.f. [75, Theorem 5]) Let* $f : \Sigma_q^n \to \Sigma_{2^{\mathcal{R}(n)}}$*, where* $\mathcal{R}(n) = o(\log\log n \cdot \log n)$*, be a labeling function for the confusable sets* $B(\boldsymbol{x}), \boldsymbol{x} \in S$*. Then there exists an integer* $a \leqslant 2^{\log \|B(\boldsymbol{x})\| + o(\log n)}$ *such that for all* $\boldsymbol{y} \in B(\boldsymbol{x})$*, we have* $f(\boldsymbol{x}) \not\equiv f(\boldsymbol{y}) \bmod a$*.*

The definitions and Theorem 102 are used in our code construction in Section 3.5.3. The construction and analysis rely on the confusable sets for the channel, discussed below.

3.5.2 Confusable sets for channels with short duplication and substitution errors

In this section, we study the size of confusable sets of input strings passing through channels with an arbitrary number of duplications and at most p substitutions. This quantity will be used to derive a Gilbert-Varshamov bound and, in the next section, to construct our error-correcting codes.

Since the duplication root is unique, and duplications and deduplications do not alter the duplication root of the input, $\mathrm{Irr}_q(n)$ is a code capable of correcting duplications. The decoding process simply removes all tandem repeats. In other words, if we append a root block, which deduplicates all repeats and produces the root of its input, to the channel with duplication errors, any irreducible



(i) DS(p) channel

(ii) DSD(p) channel

(iii) DSD(1) channel

Figure 3.6: Any error-correcting code for channel (ii) is also an error-correcting code for channel (i). The confusable set for a channel obtained by concatenating p copies of channel (iii) contains the confusable set for channel (ii).

sequence passes through this concatenated channel with no errors. This approach produces codes with the same asymptotic rate as that of [30], achieving the highest possible asymptotic rate.

Similar to [83], we extend this strategy to design codes for the channel with duplication and at most p substitution errors, denoted the DS(p) channel and shown in Figure 3.6i. Note that the duplications and substitutions can occur in any order. We take the code to be a subset of irreducible strings and find the code for a new channel obtained by concatenating a root block to the channel with duplication and substitution errors, denoted as the DSD(p) channel and shown in Figure 3.6ii. Clearly, any error-correcting code for DSD(p) is also an error-correcting code for the DS(p) channel.

We now define the confusable sets over $\mathrm{Irr}_q(n)$ for the DSD(p) channel and bound its size, which is needed to construct the code and determine its rate.

Definition 73. *For $\boldsymbol{x} \in \mathrm{Irr}_q(n)$, let*

$$B_{\mathrm{Irr}}^{\leqslant p}(\boldsymbol{x}) = \{\boldsymbol{y} \in \mathrm{Irr}_q(n) : \boldsymbol{y} \neq \boldsymbol{x}, \\ R(\mathcal{D}^{\leqslant p}(\boldsymbol{x})) \cap R(\mathcal{D}^{\leqslant p}(\boldsymbol{y})) \neq \varnothing\}$$

denote the irreducible-confusable set *of $\boldsymbol{x}$.*

Note that the DSD(1) channel can be represented as shown in Figure 3.6iii. This is because the sequence of errors consists of duplications, substitutions, more duplications, and finally all deduplications. Hence, duplications that occur after the substitutions are all deduplicated and we may equivalently assume they have not occurred. Next, observe that the confusable set for the concatenation of p DSD(1) channels contains the confusable set for a DSD(p) channel. In other words, if the same string is input to the concatenation of p DSD(1) channels and to a DSD(p) channel, the set of possible outputs of the former is a superset of the set of possible outputs of the latter. Hence, we have the following lemma.

Lemma 74. *An error-correcting code for the concatenation of p DSD(1) channels is also an error-correcting code for the DSD(p) channel.*

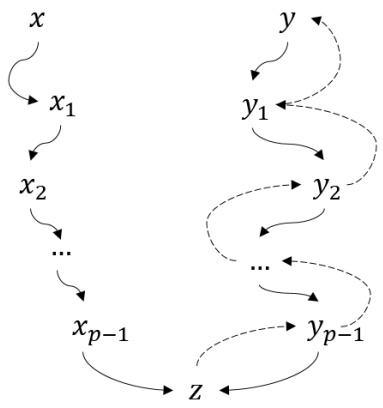


Figure 3.7: A sequence $\boldsymbol{z} = \boldsymbol{x}_p = \boldsymbol{y}_p$ that can be obtained from both $\boldsymbol{x}$ and $\boldsymbol{y}$ through channels resulting from the concatenation of p DSD(1) channels, each shown by a solid arrow. The dashed arrows represent the reverse relationships and each $\boldsymbol{y}_{i-1}$ can be obtained by passing $\boldsymbol{y}_i$ through a DSD(1) channel.

We can thus focus on this concatenated channel. The advantage of considering DSD(1) is that it is reversible in the sense that if $\boldsymbol{v}$ can be obtained from an input $\boldsymbol{u}$, then $\boldsymbol{u}$ can be obtained from the input $\boldsymbol{v}$, and this simplifies our analysis. In particular, we have $\boldsymbol{u} \in R(\mathcal{D}^{\leqslant 1}(\boldsymbol{v}))$ and $\boldsymbol{v} \in R(\mathcal{D}^{\leqslant 1}(\boldsymbol{u}))$.

Figure 3.7 shows a confusable string $\boldsymbol{z}$ obtainable from irreducible sequences $\boldsymbol{x} \in \mathrm{Irr}_q(n)$ and $\boldsymbol{y} \in B^{\leqslant p}_{\mathrm{Irr}}(\boldsymbol{x})$, after passing through p DSD(1) channels, each represented by a solid arrow. More precisely, $\boldsymbol{x}_i \in R(\mathcal{D}^{\leqslant 1}(\boldsymbol{x}_{i-1}))$ and $\boldsymbol{y}_i \in R(\mathcal{D}^{\leqslant 1}(\boldsymbol{y}_{i-1}))$, where $\boldsymbol{x} = \boldsymbol{x}_0, \boldsymbol{y} = \boldsymbol{y}_0, \boldsymbol{z} = \boldsymbol{x}_p = \boldsymbol{y}_p$. Furthermore, $\boldsymbol{y}_{i-1} \in R(\mathcal{D}^{\leqslant 1}(\boldsymbol{y}_i))$. Hence, $\boldsymbol{y}$ can be generated from $\boldsymbol{x}$ by concatenating the solid-line path from $\boldsymbol{x}$ to $\boldsymbol{z}$ and the dashed-line path from $\boldsymbol{z}$ to $\boldsymbol{y}$, i.e., $\boldsymbol{x} \to \boldsymbol{x}_1 \to \cdots \to \boldsymbol{z} \to \boldsymbol{y}_{p-1} \to \cdots \to \boldsymbol{y}$, where each $\to$ represents a DSD(1) channel. Considering the number of possibilities in each step gives the following lemma.

Lemma 75. *For $\boldsymbol{x} \in \mathrm{Irr}_q(n)$,*

$$\|B^{\leqslant p}_{\mathrm{Irr}}(\boldsymbol{x})\| \leqslant \max_{\boldsymbol{x}_i, \boldsymbol{y}_i} \prod_{i=0}^{p-1} \|R(\mathcal{D}^{\leqslant 1}(\boldsymbol{x}_i))\| \prod_{i=1}^{p} \|R(\mathcal{D}^{\leqslant 1}(\boldsymbol{y}_i))\|$$

where the maximum for $\boldsymbol{x}_i$ (resp. $\boldsymbol{y}_i$) is over sequences that can result from $\boldsymbol{x}$ (resp. $\boldsymbol{y}$) passing through the concatenation of i DSD(1) channels.

It thus suffices to find $\|R(\mathcal{D}^{\leqslant 1}(\boldsymbol{x}))\|$ for $\boldsymbol{x} \in \mathrm{Irr}_q(*)$. As

$$\begin{aligned} \|R(\mathcal{D}^{\leqslant 1}(\boldsymbol{x}))\| &\leqslant \|R(\mathcal{D}^{1}(\boldsymbol{x}))\| + \|R(\mathcal{D}(\boldsymbol{x}))\| \\ &= \|R(\mathcal{D}^{1}(\boldsymbol{x}))\| + 1, \end{aligned}$$

we find an upper bound on $\|R(\mathcal{D}^1(\boldsymbol{x}))\|$, in Lemma 77, using the following lemma from [83].

Lemma 76. *[83, Lemma 3] Let $\boldsymbol{x}$ be any string of length at least 5 and $\boldsymbol{x}' \in \mathcal{D}(\boldsymbol{x})$. For any decomposition of $\boldsymbol{x}$ as*

$$\boldsymbol{x} = \ \boldsymbol{r}\ ab\ c\ de\ \boldsymbol{s},$$

for $a, b, c, d, e \in \Sigma_q$ and $\boldsymbol{r}, \boldsymbol{s} \in \Sigma_q^*$, there is a decomposition of $\boldsymbol{x}'$ as

$$\boldsymbol{x}' = \boldsymbol{u}\ ab\ \boldsymbol{w}\ de\ \boldsymbol{v}$$

such that $\boldsymbol{u}, \boldsymbol{w}, \boldsymbol{v} \in \Sigma_q^*$, $\boldsymbol{u}ab \in \mathcal{D}(\boldsymbol{r}ab)$, $ab\boldsymbol{w}de \in \mathcal{D}(abcde)$, and $de\boldsymbol{v} \in \mathcal{D}(de\boldsymbol{s})$.

Lemma 77. *For an irreducible string $\boldsymbol{x} \in \Sigma_q^n$,*

$$\|R(\mathcal{D}^1(\boldsymbol{x}))\| \leqslant n \max_{\boldsymbol{t} \in \Sigma_q^5} \|R(\mathcal{D}^1(\boldsymbol{t}))\|.$$

Proof. Given an irreducible string $\boldsymbol{x} \in \mathrm{Irr}_q(n)$, let $\boldsymbol{x}' \in \mathcal{D}(\boldsymbol{x})$ be obtained from $\boldsymbol{x}$ through duplications and $\boldsymbol{x}''$ obtained from $\boldsymbol{x}'$ by a substitution. For a given $\boldsymbol{x}$, $\|R(\mathcal{D}^1(\boldsymbol{x}))\|$ equals the number of possibilities for $R(\boldsymbol{x}'')$ as $\boldsymbol{x}''$ varies. Note that duplications that occur after the substitution do not affect the root. So we have assumed that the substitution is the last error before the root is found.

Decompose $\boldsymbol{x}$ as $\boldsymbol{x} = \boldsymbol{r}abcde\boldsymbol{s}$ with $\boldsymbol{r}, \boldsymbol{s} \in \mathrm{Irr}_q(*)$ and $a, b, c, d, e \in \Sigma_q$, so that the substituted symbol in $\boldsymbol{x}'$ is a copy of c. Note that if $|\boldsymbol{x}| < 5$ or if a copy of one of its first two symbols or its last two symbols is substituted, then we can no longer write $\boldsymbol{x}$ as described. To avoid considering these cases separately, we may append two dummy symbols to the beginning of $\boldsymbol{x}$ and two dummy symbols to the end of $\boldsymbol{x}$, where the four dummy symbols are distinct and do not belong to Σ_q, and prove the result for this new string. Since these dummy symbols do not participate in any duplication, substitution, or deduplication events, the proof is also valid for the original $\boldsymbol{x}$.

By Lemma 76, we can write

$$\begin{aligned} \boldsymbol{x} &= \ \boldsymbol{r}\ ab\ c\ de\ \boldsymbol{s} \\ \boldsymbol{x}' &= \boldsymbol{u}\ ab\ \boldsymbol{w}\ de\ \boldsymbol{v}, \\ \boldsymbol{x}'' &= \boldsymbol{u}\ ab\ \boldsymbol{z}\ de\ \boldsymbol{v}, \end{aligned} \tag{3.12}$$

where $\boldsymbol{u}ab \in \mathcal{D}(\boldsymbol{r}ab)$, $ab\boldsymbol{w}de \in \mathcal{D}(abcde)$, $de\boldsymbol{v} \in \mathcal{D}(de\boldsymbol{s})$, and $\boldsymbol{z}$ is obtained from $\boldsymbol{w}$ by substituting a copy of c. From (3.12), $R(\boldsymbol{x}'') = R(\boldsymbol{r}R(ab\boldsymbol{z}de)\boldsymbol{s})$, where $R(ab\boldsymbol{z}de)$ starts with ab and ends with de (which may fully or partially overlap).

To determine $\|R(\mathcal{D}^1(\boldsymbol{x}))\|$, we count the number of possibilities for $R(\boldsymbol{x}'')$ as $\boldsymbol{x}''$ varies. Considering the decomposition of $\boldsymbol{x}''$ into $\boldsymbol{u}ab\boldsymbol{z}de\boldsymbol{v}$ given in (3.12), we note that if $R(ab\boldsymbol{z}de)$ is given, then $R(\boldsymbol{x}'') = R(\boldsymbol{r}R(ab\boldsymbol{z}de)\boldsymbol{s})$ is uniquely determined. So to find an upper bound, it suffices to count the number of possibilities for $R(ab\boldsymbol{z}de)$. We thus have

$$\|R(\mathcal{D}^1(\boldsymbol{x}))\| \leqslant \sum \|\{R(ab\boldsymbol{z}de) : ab\boldsymbol{z}de \in \mathcal{D}^1(abcde)\}\|,$$

where the sum is over the choices of c in $\boldsymbol{x}$, or equivalently the decompositions of $\boldsymbol{x}$ into $\boldsymbol{r}abcde\boldsymbol{s}$,

in (3.12). As there are n choices for c, we have

$$\|R(\mathcal{D}^1(\boldsymbol{x}))\| \leqslant n \max_{\boldsymbol{t}\in\Sigma_q^5} \|R(\mathcal{D}^1(\boldsymbol{t}))\|.$$

□

The next lemma provides a bound on $\|R(\mathcal{D}^1(\boldsymbol{t}))\|$ for $\boldsymbol{t} \in \Sigma_q^5$ by identifying the "worst case". The proof is given in Appendix A.

Lemma 78. *Given $q \geqslant 3$, we have*

$$\max_{\boldsymbol{t}\in\Sigma_q^5} \|R(\mathcal{D}^1(\boldsymbol{t}))\| \leqslant \|R(\mathcal{D}^1(01234))\|,$$

where $\mathcal{D}^1(01234) \subseteq \Sigma_{q+4}^*$ (the substituted symbol can be replaced with another symbol from Σ_{q+4}).

As shown in [29], $\mathcal{D}(01234)$ is a regular language whose words can be described as paths from 'Start' to S_9 in the finite automaton given in Figure 3.2. Then $\mathcal{D}^1(01234)$ is equivalent to the set of paths from 'Start' to S_9 but with the label on one edge substituted. We will use this observation to bound $\|R(\mathcal{D}^1(01234))\|$ in Lemma 80. The next lemma establishes a symmetric property of the automaton that will be useful in Lemma 80. Lemma 79 is proved by showing that there is a bijective function $h : U \to V$ between U and V and between $R(U)$ and $R(V)$. Specifically, for $\boldsymbol{u} = u_1 \cdots u_n$, we let $\boldsymbol{v} = h(\boldsymbol{u}) = \bar{u}_n \bar{u}_{n-1} \ldots \bar{u}_1$, where for $a \in \{0, 1, 2, 3, 4\}$, $\bar{a} = 4 - a$.

Lemma 79. *Let U and V be the sets of labels of all paths from Start to any state and from any state to S_9, respectively, in the finite automaton of Figure 3.2. Then $\|U\| = \|V\|$ and $\|R(U)\| = \|R(V)\|$.*

Proof. Define $h(a) = 4 - a$ for $a \in \Sigma_5$ and $h(\boldsymbol{u}) = h(u_n)h(u_{n-1}) \cdots h(u_1)$ for $\boldsymbol{u} \in \Sigma_5^n$. Furthermore, for $S \subseteq \Sigma_5^*$, define $h(S) = \{h(\boldsymbol{u}) : \boldsymbol{u} \in S\}$. Note that h is its own inverse. We claim that h has the following properties, to be proved later:

1. For $\boldsymbol{s}, \boldsymbol{t} \in \Sigma_5^*$, $\boldsymbol{s}$ is a prefix of $\boldsymbol{t}$ if and only if $h(\boldsymbol{s})$ is a suffix of $h(\boldsymbol{t})$.
2. For $\boldsymbol{t} \in \Sigma_5^*$, $\mathcal{D}(h(\boldsymbol{t})) = h(\mathcal{D}(\boldsymbol{t}))$.
3. For $S \subseteq \Sigma_5^*$, $R(h(S)) = h(R(S))$.

By definition, if $\boldsymbol{u} \in U$ then $\boldsymbol{u}$ is a prefix of some $\boldsymbol{x} \in \mathcal{D}(01234)$. Then, by Property 1, $h(\boldsymbol{u})$ is a suffix of $h(\boldsymbol{x})$. By setting $\boldsymbol{t} = 01234$, it follows from Property 2 that $\mathcal{D}(01234) = h(\mathcal{D}(01234))$, and thus $h(\boldsymbol{x}) \in \mathcal{D}(01234)$. Hence, $h(\boldsymbol{u})$ is in V. Similarly, we can show that if $\boldsymbol{v} \in V$, then $h(\boldsymbol{v}) \in U$. As h is its own inverse, we have $V = h(U)$ and $\|U\| = \|V\|$. Applying Property 3 with $S = U$ yields $R(V) = h(R(U))$ and $\|R(V)\| = \|R(U)\|$.

We now prove Properties 1-3. Property 1 follows from the definition of h. Property 2 follows from the observation that if $\boldsymbol{x}'$ is obtained from $\boldsymbol{x}$ via a duplication, then $h(\boldsymbol{x}')$ can be obtained from $h(\boldsymbol{x})$ via a duplication, i.e., the relationship represented by h is maintained under duplication. To prove Property 3, it suffices to show that $R(h(\boldsymbol{t})) = h(R(\boldsymbol{t}))$ for $\boldsymbol{t} \in \Sigma_5^*$, which holds as h is maintained under deduplication. □

Lemma 80. *For $\hat{q} \geqslant 5$ and $\mathcal{D}^1(01234) \subseteq \Sigma_{\hat{q}}^*$, where the substitution replaces a symbol with any symbol from $\Sigma_{\hat{q}}$, we have*

$$\|R(\mathcal{D}^1(01234))\| \leqslant 22^2(\hat{q}-1).$$

Proof. Based on [29], recall that $\mathcal{D}(01234)$ is a regular language whose words can be described as paths from 'Start' to S_9 in the finite automaton given in Figure 3.2, where the word associated with each path is the sequence of the edge labels. Let $\boldsymbol{x}' \in \mathcal{D}(01234)$ and let $\boldsymbol{x}''$ be generated from $\boldsymbol{x}'$ by a substitution. Assume $\boldsymbol{x}' = \boldsymbol{u}w\boldsymbol{v}$ and $\boldsymbol{x}'' = \boldsymbol{u}\hat{w}\boldsymbol{v}$, where $\boldsymbol{u}, \boldsymbol{v} \in \Sigma_5^*, w \in \Sigma_5$ and $\hat{w} \in \Sigma_{\hat{q}} \setminus \{w\}$. So there are $\hat{q}-1$ choices for $\hat{w}$. The string $\boldsymbol{u}$ represents a path from 'Start' to some state s_u and the string $\boldsymbol{v}$ represents a path from some state s_v to S_9 in the automaton, where there is an edge with label w from s_u to s_v.

As $\boldsymbol{x}'' = \boldsymbol{u}\hat{w}\boldsymbol{v}$, we have $R(\boldsymbol{x}'') = R(R(\boldsymbol{u})\hat{w}R(\boldsymbol{v}))$, where $R(\boldsymbol{u})$ is an irreducible string represented by a path from "Start" to state s_u, and $R(\boldsymbol{v})$ is an irreducible string represented by a path from s_v to S_9. Define U and V as in Lemma 79. We thus have $\|R(\mathcal{D}^1(\boldsymbol{x}))\| \leqslant ||R(U)|| \times (\hat{q}-1) \times ||R(V)|| = ||R(U)||^2 \times (\hat{q}-1)$. By inspection, we can show that

$$\begin{aligned} R(U) = \{&\Lambda, 0, 01, 01201, 012, 0120, 010, 012010,\\ &0121, 01202, 0123, 01232, 01231, 012313, 012312,\\ &0123121, 01234, 012343, 012342, 0123424,\\ &0123423, 01234232\}, \end{aligned}$$

and hence $||R(U)|| = 22$, completing the proof. □

Theorem 81. *For an irreducible string $\boldsymbol{x} \in \Sigma_q^n$, with $q \geqslant 3$,*

$$\|R(\mathcal{D}^{\leqslant 1}(\boldsymbol{x}))\| \leqslant 968nq + 1.$$

Proof. From Lemmas 77, 78, and 80, it follows that $\|R(\mathcal{D}^1(\boldsymbol{x}))\| \leqslant 22^2 n(\hat{q}-1) \leqslant 2q \cdot 22^2 n = 968nq$ with $\hat{q} = q+4$. Furthermore, $\|R(\mathcal{D}^{\leqslant 1}(\boldsymbol{x}))\| \leqslant \|R(\mathcal{D}^1(\boldsymbol{x}))\| + 1$. □

We can now use Theorem 81 along with Lemma 75, to find a bound on $\|B_{\text{Irr}}^{\leqslant p}(\boldsymbol{x})\|$. To do so, we need to bound the size of $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ shown in Figure 3.7, for which the following theorem is of use. The theorem is a direct extension of [83, Theorem 5] and thus requires no proof. An example demonstrating the theorem is given in Example 61.

Theorem 82 (c.f.[83, Theorem 5])**.** *Given strings $\boldsymbol{x} \in \Sigma_q^n$ and $\boldsymbol{v} \in \mathcal{D}^{\leqslant p}(\boldsymbol{x})$, $R(\boldsymbol{v})$ can be obtained from $R(\boldsymbol{x})$ by at most p $\mathcal{L}$-substring edits, where $\mathcal{L} = 17$.*

It follows from the theorem that for $1 \leqslant i \leqslant p$,

$$|\boldsymbol{x}_i| \leqslant n + p\mathcal{L}, \ |\boldsymbol{y}_i| \leqslant n + p\mathcal{L}.$$

The next theorem then follows from Lemmas 75 and 81.

Theorem 83. *Let $\boldsymbol{x} \in \mathrm{Irr}_q(n) \subseteq \Sigma_q^n$ be an irreducible string of length n with $q \geqslant 3$. The irreducible-confusable set $B_{\mathrm{Irr}}^{\leqslant p}(\boldsymbol{x})$ of $\boldsymbol{x}$ satisfies*

$$\|B_{\mathrm{Irr}}^{\leqslant p}(\boldsymbol{x})\| \leqslant (968q(n + p\mathcal{L}) + 1)^{2p}.$$

The size of the confusable sets will be used for our code construction. It also allows us to derive a Gilbert-Varshamov (GV) bound, as follows.

Theorem 84. *There exists a code of length n capable of correcting any number of duplications and at most p substitutions with size at least*

$$\frac{\|\mathrm{Irr}_q(n)\|}{(968q(n + p\mathcal{L}) + 1)^{2p}}.$$

We will show in Lemma 95 that the size of the code with the highest asymptotic rate for correcting duplications only is essentially $\|\mathrm{Irr}_q(n)\|$. Assuming that p and q are constants, this GV bound shows that a code exists for additionally correcting up to p substitution errors with extra redundancy of approximately $2p \log_q n$ symbols. The two constructions presented in the next section have extra redundancies of $4p \log_q n$ and $8p \log_q n$, which are only small constant factors away from this existential bound.

3.5.3 Low-redundancy error-correcting codes

As stated in Section 5.3, our code for correcting duplications and substitutions is a subset of irreducible strings of a given length. In this section, we construct this subset by applying the syndrome compression technique [75], where we will make use of the size of the irreducible-confusable set $\|B_{\mathrm{Irr}}^{\leqslant p}(\boldsymbol{x})\|$ derived in Section 5.3. In this section, unless otherwise stated, we assume that both $q \geqslant 4$ and p are constant.

We begin by presenting the code constructions for correcting duplications and *substitutions* in Subsection 3.5.3, assuming the existence of appropriate labeling functions used to produce the syndrome information and an auxiliary error-correcting code used to protect it. The labeling functions will be discussed in Subsection 3.5.6, while the auxiliary ECC is presented in Section 3.4. In Subsection 3.5.5, we show that the proposed codes can in fact correct duplications and *edits*. The redundancy of the codes and the computational complexities of their encoding and decoding are discussed in Subsections 3.5.7 and 3.5.8, respectively.

We first present a code in Construction 85 that assumes an error-free side channel is available, where the length of the sequence passing through the side channel is logarithmic in the length of the sequence passing through the main channel. We then present the main result of this section, Construction 90, which does not make such an assumption and is intended for a single noisy channel. Construction 85 helps motivate certain components of Construction 90 and make its proof of correctness more clear. In addition, it may have potential practical applications. For example, in a DNA storage system, metadata of the data stored on DNA may be stored on silicon-based devices such as disk or flash. Due to the maturity of these technologies, they may provide a nearly error-free

channel, suitable for storing a small amount of side information.

a) Channels with error-free side channels

In the construction below, $\boldsymbol{x}$ is transmitted through the noisy channel, while $\boldsymbol{r}$, which encodes the information $(a, f(\boldsymbol{x}) \bmod a)$ is transmitted through an error-free channel.

Construction 85. *Let n, p, q be positive integers. Furthermore, let f be a (labeling) function and, for each $\boldsymbol{x} \in \mathrm{Irr}_q(n)$, $a_{\boldsymbol{x}}$ be a positive integer, such that for any $\boldsymbol{y} \in B_{\mathrm{Irr}}^{\leqslant p}(\boldsymbol{x})$, $f(\boldsymbol{x}) \not\equiv f(\boldsymbol{y}) \bmod a_{\boldsymbol{x}}$. Define*

$$\mathcal{C}_n^A = \{(\boldsymbol{x}, \boldsymbol{r}) : \boldsymbol{x} \in \mathrm{Irr}_q(n), \boldsymbol{r} = (a_{\boldsymbol{x}}, f(\boldsymbol{x}) \bmod a_{\boldsymbol{x}})\},$$

where $\boldsymbol{r}$ is assumed to be the q-ary representation of $(a_{\boldsymbol{x}}, f(\boldsymbol{x}) \bmod a_{\boldsymbol{x}})$.

We consider the length of this code to be $N = n + |\boldsymbol{r}|$. As will be observed in (3.13), $|\boldsymbol{r}| = O(\log_q n)$ and so the sequence carried by the side channel is logarithmic in length. Recall that the existence of the labeling functions is discussed in Subsection 3.5.6.

Theorem 86. *The code in Construction 85, assuming the labeling function f and $a_{\boldsymbol{x}}$ (for each $\boldsymbol{x} \in \mathrm{Irr}_q(n)$) exist, can correct any number of duplications and at most p substitutions applied to $\boldsymbol{x}$, provided that $\boldsymbol{r}$ is transmitted through an error-free channel.*

Proof. Let the retrieved word from storing $\boldsymbol{x}$ be $\boldsymbol{v} \in R(\mathcal{D}^{\leqslant p}(\boldsymbol{x}))$. Note that $a_{\boldsymbol{x}}$ and $f(\boldsymbol{x}) \bmod a_{\boldsymbol{x}}$ can be recovered error-free from $\boldsymbol{r}$. By definition, for all $\boldsymbol{y} \neq \boldsymbol{x}$ that could produce the same $\boldsymbol{v}$, we have $\boldsymbol{y} \in B_{\mathrm{Irr}}^{\leqslant p}(\boldsymbol{x})$. But then, $f(\boldsymbol{y}) \not\equiv f(\boldsymbol{x}) \bmod a_{\boldsymbol{x}}$, and so we can determine $\boldsymbol{x}$ by exhaustive search. □

b) Channels with no side channels

To better illustrate the construction with no side channels, let us first observe what the issues are with simply concatenating $\boldsymbol{x}$ and $\boldsymbol{r}$ and forming codewords of the form $\boldsymbol{xr}$.

- The code in Construction 85 relies on a sequence $\boldsymbol{v} \in R(\mathcal{D}^{\leqslant p}(\boldsymbol{x}))$ but if $\boldsymbol{xr}$ is stored, the output of the channel is a sequence $\boldsymbol{w} \in R(\mathcal{D}^{\leqslant p}(\boldsymbol{xr}))$. As the boundary between $\boldsymbol{x}$ and $\boldsymbol{r}$ becomes unclear after duplication and substitution errors, it is difficult to find $\boldsymbol{v} \in R(\mathcal{D}^{\leqslant p}(\boldsymbol{x}))$ from $\boldsymbol{w} \in R(\mathcal{D}^{\leqslant p}(\boldsymbol{xr}))$. To address this, instead of finding $\boldsymbol{v}$, we find a sufficiently long prefix, as discussed in Lemma 87. This will also require us to modify the labeling function.

- The decoding process requires the information encoded in $\boldsymbol{r}$, which is now subject to errors. We will address this by using a high-redundancy code that can protect this information, introduced in Lemma 88 and discussed in detail in Subsection 3.5.4.

- The codewords need to be irreducible. This is discussed in Lemma 89.

For integers p, j, denote by $\mathcal{D}_{\leqslant j}^{\leqslant p}(\boldsymbol{x})$ the set of strings that can be obtained by deleting a suffix of length at most j from some $\boldsymbol{v} \in R(\mathcal{D}^{\leqslant p}(\boldsymbol{x}))$. Note that $\mathcal{D}_{\leqslant j}^{\leqslant p}(\boldsymbol{x}) \subseteq \mathrm{Irr}_q(*)$.

Lemma 87. *Let $\boldsymbol{x}$ be an irreducible string of length n and $\boldsymbol{r}$ any string such that $\boldsymbol{xr}$ is irreducible. Let $\boldsymbol{w} \in R(\mathcal{D}^{\leqslant p}(\boldsymbol{xr}))$ and $\boldsymbol{s}$ be the prefix of $\boldsymbol{w}$ of length $n - p\mathcal{L}$. Then $\boldsymbol{s} \in \mathcal{D}^{\leqslant p}_{\leqslant 2p\mathcal{L}}(\boldsymbol{x})$.*

Proof. Based on Theorem 82, $\boldsymbol{w}$ can be considered as being generated from $\boldsymbol{xr}$ by at most p $\mathcal{L}$-substring edits. Let j be the last symbol of $\boldsymbol{x}$ not affected by a substring edit (i.e., it is not deleted by a substring edit, but it may be shifted). Suppose $t \leqslant p$ substring edits occur before x_j and at most $p - t$ after x_j. Then, $j \in [n - (p - t)\mathcal{L}, n]$. The symbol x_j appears as the ith symbol of $\boldsymbol{w}$ for some $i \in [j - t\mathcal{L}, j + t\mathcal{L}]$. Then, $\boldsymbol{w}_{[i]} \in R(\mathcal{D}^t(\boldsymbol{x}_{[j]}))$. It follows that $\boldsymbol{v} \in R(\mathcal{D}^t(\boldsymbol{x}))$ for $\boldsymbol{v} = \boldsymbol{w}_{[i]}\boldsymbol{x}_{[j+1,n]}$. As $i \geqslant j - t\mathcal{L}$ and $j \geqslant n - (p - t)\mathcal{L}$, we have $n - p\mathcal{L} \leqslant i$. Hence, $\boldsymbol{s} = \boldsymbol{w}_{[n-p\mathcal{L}]}$ is a prefix of $\boldsymbol{w}_{[i]}$ and thus also a prefix of $\boldsymbol{v}$. Specifically, $\boldsymbol{s}$ can be obtained from $\boldsymbol{v}$ by a suffix deletion of length

$$\begin{aligned} |\boldsymbol{v}| - (n - p\mathcal{L}) &= i + (n - j) - (n - p\mathcal{L}) \\ &\leqslant n + t\mathcal{L} + (p - t)\mathcal{L} - (n - p\mathcal{L}) \\ &= 2p\mathcal{L}. \end{aligned}$$

As $\boldsymbol{v} \in \mathcal{D}^{\leqslant p}(\boldsymbol{x})$, we have $\boldsymbol{s} \in \mathcal{D}^{\leqslant p}_{\leqslant 2p\mathcal{L}}(\boldsymbol{x})$. □

By choosing the first $n - p\mathcal{L}$ elements of $\boldsymbol{w} \in R(\mathcal{D}^{\leqslant p}(\boldsymbol{xr}))$, we find $\boldsymbol{s} \in \mathcal{D}^{\leqslant p}_{\leqslant 2p\mathcal{L}}(\boldsymbol{x})$, which is a function of only $\boldsymbol{x}$ rather than $\boldsymbol{xr}$. But in doing so, we have introduced an additional error, namely deleting a suffix of length at most $2p\mathcal{L}$. As a result, we need to replace the labeling function f with a stronger labeling function f' that, in addition to handling both substitutions and duplications, can handle deleting a suffix of $\boldsymbol{x}$. More precisely, f' is a labeling function for the confusable set

$$\begin{aligned} B^{\leqslant p, \leqslant 2p\mathcal{L}}_{\mathrm{Irr}}(\boldsymbol{x}) = \{&\boldsymbol{y} \in \mathrm{Irr}_q(n) : \\ &\boldsymbol{y} \neq \boldsymbol{x}, \mathcal{D}^{\leqslant p}_{\leqslant 2p\mathcal{L}}(\boldsymbol{x}) \cap \mathcal{D}^{\leqslant p}_{\leqslant 2p\mathcal{L}}(\boldsymbol{y}) \neq \varnothing\}. \end{aligned}$$

The details of determining f' will be discussed in Section 3.5.6. Assuming the existence of the labeling function, $\boldsymbol{r}$ encodes $(a'_{\boldsymbol{x}}, f'(\boldsymbol{x}) \bmod a'_{\boldsymbol{x}})$, where for $\boldsymbol{x} \in \mathrm{Irr}_q(\boldsymbol{x})$, $a'_{\boldsymbol{x}}$ is chosen such that

$$f'(\boldsymbol{x}) \not\equiv f'(\boldsymbol{y}) \bmod a'_{\boldsymbol{x}}, \text{ for all } \boldsymbol{y} \in B^{\leqslant p, \leqslant 2p\mathcal{L}}_{\mathrm{Irr}}(\boldsymbol{x}).$$

To address the second difficulty raised above, i.e., protecting the information encoded in $\boldsymbol{r}$, we use an auxiliary high-redundancy code given in Section 3.4. The following lemma, which is proved in Subsection 3.5.4, provides an encoder for this code.

Lemma 88. *Let $\boldsymbol{\sigma} = 01020$. There exists an encoder $\mathcal{E}_1 : \Sigma_2^L \to \mathrm{Irr}_q(L')$ such that i) $\boldsymbol{\sigma}\mathcal{E}_1(\boldsymbol{u}) \in \mathrm{Irr}_q(*)$ and ii) for any string $\boldsymbol{x} \in \mathrm{Irr}_q(*)$ with $\boldsymbol{x\sigma}\mathcal{E}_1(\boldsymbol{u}) \in \mathrm{Irr}_q(*)$, we can recover $\boldsymbol{u}$ from any $\boldsymbol{w} \in R(\mathcal{D}^{\leqslant p}(\boldsymbol{x\sigma}\mathcal{E}_1(\boldsymbol{u})))$. Asymptotically, $L' \leqslant \frac{L}{\log(q-2)}(1 + o(1))$.*

We use $\mathcal{E}_1(a'_{\boldsymbol{x}}, f'(\boldsymbol{x}) \bmod a'_{\boldsymbol{x}})$ to denote $\mathcal{E}_1(\boldsymbol{u})$, where $\boldsymbol{u}$ is a binary sequence representing the pair $(a'_{\boldsymbol{x}}, f'(\boldsymbol{x}) \bmod a'_{\boldsymbol{x}})$. For $\boldsymbol{x} \in \mathrm{Irr}_q(n)$, by letting $\boldsymbol{r} = \mathcal{E}_1(a'_{\boldsymbol{x}}, f'(\boldsymbol{x}) \bmod a'_{\boldsymbol{x}})$, we can construct codewords of the form $\boldsymbol{x\sigma r}$. But such codewords would not necessarily be irreducible. Irreducibility can be ensured by adding a buffer $\boldsymbol{b}_{\boldsymbol{x}}$ between $\boldsymbol{x}$ and $\boldsymbol{\sigma r}$, as described by the next lemma.

Lemma 89. *For $q \geqslant 3$ and any irreducible string $\boldsymbol{x}$ over Σ_q, there is a string $\boldsymbol{b_x}$ of length c_q such that $\boldsymbol{x b_x \sigma}$ is irreducible. Furthermore, $c_3 = 13$, $c_4 = 7$, $c_5 = 6$, and $c_q = 5$ for $q \geqslant 6$.*

Before proving Lemma 115, we recall from [30] that $\mathrm{Irr}_q(*)$ is a regular language whose graph $G_q = (V_q, \xi_q)$ is a subgraph of the De Bruijn graph. The vertex set V_q consists of 5-tuples $a_1a_2a_3a_4a_5 \in \mathrm{Irr}_q(5)$ that do not have any repeats (of length at most 4). There is an edge from $a_1a_2a_3a_4a_5 \to a_2a_3a_4a_5a_6$ if $a_1a_2a_3a_4a_5a_6$ belongs to $\mathrm{Irr}_q(6)$. The label for this edge is a_6. The label for a path is the 5-tuple representing its starting vertex concatenated with the labels of the subsequent edges. The proof below is similar to that of [83, Theorem 15] and is presented here for completeness.

Proof. Given $\boldsymbol{x} \in \mathrm{Irr}_q(n)$ and $q \geqslant 3$, $\boldsymbol{x}$ can be represented by a path over the graph G_q, ending at the vertex $\boldsymbol{x}_{[n-4:n]}$. Furthermore, $\boldsymbol{\sigma} = 01020$ can be considered as a vertex in G_q since $\boldsymbol{\sigma} \in \mathrm{Irr}_q(5)$. Let us assume for the moment that $q \geqslant 6$. Based on [83, Lemma 14], each vertex has at least $q-2$ outgoing edges. So from each vertex, there is at least one outgoing edge whose label is equal to either $3, 4$, or 5. So, starting from $\boldsymbol{x}_{[n-4:n]}$, we may arrive at some vertex with label $\boldsymbol{b_x} \in \{3,4,5\}^5$ in 5 steps. Furthermore, $\boldsymbol{b_x \sigma}$ is irreducible as both $\boldsymbol{b_x}$ and $\boldsymbol{\sigma}$ are irreducible and have no symbols in common. Hence, there is a path of length 5 from $\boldsymbol{b_x}$ to $\boldsymbol{\sigma}$ in G_q. So there is a path in G_q with label $\boldsymbol{x b_x \sigma}$, implying that $\boldsymbol{x b_x \sigma}$ is irreducible. We further have $c_q = |\boldsymbol{b_x}| = 5$. For $q \in \{3,4,5\}$, we have verified computationally that, for any choice of $\boldsymbol{x}_{[n-4:n]}$, there exists a path from $\boldsymbol{x}_{[n-4:n]}$ to $\boldsymbol{\sigma}$ of length $c_q + 5$, with the value of c_q as given in the lemma. Denoting the label of this path as $\boldsymbol{b_x \sigma}$ gives us the sequence $\boldsymbol{b_x}$ of length c_q, with $\boldsymbol{x b_x \sigma}$ being irreducible. □

The lemma implies that $\boldsymbol{x b_x \sigma r}$ is irreducible. This is because any substring of length at most 6 is either in $\boldsymbol{x b_x \sigma}$ or in $\boldsymbol{\sigma r}$ but cannot span both as $|\boldsymbol{\sigma}| = 5$. But $\boldsymbol{x b_x \sigma}$ and $\boldsymbol{\sigma r}$ are both irreducible, as shown in Lemma 115 and Lemma 88.i, respectively.

We are now ready to present the code construction.

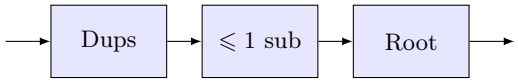
Construction 90. *Let f' be a labeling function for the confusable sets $B_{\mathrm{Irr}}^{\leqslant p, \leqslant 2p\mathcal{L}}(\boldsymbol{x}), \boldsymbol{x} \in \mathrm{Irr}_q(n)$. Furthermore, for each $\boldsymbol{x}$, let $a'_{\boldsymbol{x}}$ be an integer such that $f'(\boldsymbol{x}) \not\equiv f'(\boldsymbol{y}) \bmod a'_{\boldsymbol{x}}$ for $\boldsymbol{y} \in B_{\mathrm{Irr}}^{\leqslant p, \leqslant 2p\mathcal{L}}(\boldsymbol{x})$. Let*

$$\mathcal{C}_n^B = \{\boldsymbol{x b_x \sigma r} : \boldsymbol{x} \in \mathrm{Irr}_q(n), \boldsymbol{r} = \mathcal{E}_1(a'_{\boldsymbol{x}}, f'(\boldsymbol{x}) \bmod a'_{\boldsymbol{x}})\}.$$

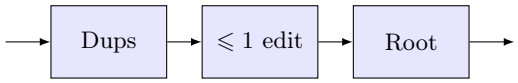
Note that for simplicity, we index the code by the length of $\boldsymbol{x}$ rather than the length of the codewords $\boldsymbol{x b_x \sigma r}$, i.e., n in $\mathcal{C}_n^B$ refers to the length of $\boldsymbol{x}$. The length of $\boldsymbol{r}$ is discussed in Subsection 3.5.7 below.

Theorem 91. *The code in Construction 90 can correct any number of short duplications and at most p substitutions.*

Proof. Let the retrieved word be $\boldsymbol{w} \in R(\mathcal{D}^{\leqslant p}(\boldsymbol{x b_x \sigma r}))$. From Lemma 88, given $\boldsymbol{w}$, we can find $a'_{\boldsymbol{x}}$ and $f'(\boldsymbol{x}) \bmod a'_{\boldsymbol{x}}$. By Lemma 115, $\boldsymbol{x b_x \sigma r}$ is irreducible. Then, by Lemma 87, the $(n - p\mathcal{L})$-prefix of $\boldsymbol{w}$, denoted $\boldsymbol{s}$, satisfies $\boldsymbol{s} \in \mathcal{D}^{\leqslant p}_{\leqslant 2p\mathcal{L}}(\boldsymbol{x})$. By definition, for all $\boldsymbol{y} \neq \boldsymbol{x}$ that could produce the same $\boldsymbol{s}$, we have $\boldsymbol{y} \in B_{\mathrm{Irr}}^{\leqslant p, \leqslant 2p\mathcal{L}}(\boldsymbol{x})$. But then, $f'(\boldsymbol{y}) \not\equiv f'(\boldsymbol{x}) \bmod a'_{\boldsymbol{x}}$, and so we can determine $\boldsymbol{x}$ by exhaustive search. □



(i) DSD(1) channel



(ii) DED(1) channel

Figure 3.8: Any error-correcting code for channel (i) is also an error-correcting code for channel (ii).

3.5.4 Proof of Lemma 88

To simplify the proof, instead of directly proving Lemma 88, we prove the following lemma, which essentially reverses the sequences in Lemma 88. Since both duplication and deduplication are symmetric operations, the lemmas are equivalent.

Lemma 92. *Let* $\boldsymbol{\sigma} = 01020$. *There exists an encoder* $\mathcal{E}_1 : \Sigma_2^L \to \mathrm{Irr}_q(L')$ *such that i)* $\mathcal{E}_1(\boldsymbol{u})\boldsymbol{\sigma} \in \mathrm{Irr}_q(*)$ *and ii) for any string* $\boldsymbol{x} \in \mathrm{Irr}_q(*)$ *with* $\mathcal{E}_1(\boldsymbol{u})\boldsymbol{\sigma}\boldsymbol{x} \in \mathrm{Irr}_q(*)$, *we can recover* $\boldsymbol{u}$ *from any* $\boldsymbol{w} \in R(\mathcal{D}^{\leqslant p}(\mathcal{E}_1(\boldsymbol{u})\boldsymbol{\sigma}\boldsymbol{x}))$. *Asymptotically,* $L' \leqslant L/\log(q-2)(1+o(1))$.

Proof. Let $\boldsymbol{v} = \mathcal{E}_1(\boldsymbol{u})$ and $\boldsymbol{w} \in R(\mathcal{D}^{\leqslant p}(\boldsymbol{v}\boldsymbol{\sigma}\boldsymbol{x}))$. Furthermore, let $\boldsymbol{s}$ be $|\boldsymbol{v}| - p\mathcal{L}$-prefix of $\boldsymbol{w}$. By Lemma 87, we have $\boldsymbol{s} \in \mathcal{D}^{\leqslant p}_{\leqslant 2p\mathcal{L}}(\boldsymbol{v})$. So $\boldsymbol{s}$ can be obtained from $\boldsymbol{v}$ through at most $3p$ $\mathcal{L}$-substring edits. So if we let $\mathcal{E}_1$ be an encoder for $\mathcal{C}_E$ designed to correct $3p$ substitution errors and an infinite number of duplications, we can recover $\boldsymbol{u}$ from $\boldsymbol{s}$. The rate of this encoder is lower bounded by $\log(q-2)(1+o(1))$. □

3.5.5 Extension to edit errors

We now show that the codes in Constructions 85 and 90 are able to correct an arbitrary number of duplications and at most p *edit* errors, where an edit error may be a deletion, an insertion, or a substitution.

Define the DED(1) and DED(p) channels analogously to the DSD(1) and DSD(p) channels by replacing substitutions with edit errors. Any error-correcting code for a concatenation of p DED(1) channels is also an error-correcting code for DED(p).

Additionally, any error-correcting code for a DSD(1) channel is also an error-correcting code for the DED(1) channel. This is because any input-output pair $(\boldsymbol{x}, \boldsymbol{y})$ for DED(1), shown in Figure 3.8ii, is also an input-output pair for the DSD(1) channel, shown in Figure 3.8i. This claim is proved in [83, Corollary 12], where it was shown that a deletion can be represented as a substitution and a deduplication, e.g., $abc \to ac$ as $abc \to aac \to ac$, and an insertion as a duplication and a substitution, e.g., $abc \to abdc$ as $abc \to abbc \to abdc$.

Since $\mathcal{C}^A$ and $\mathcal{C}^B$ can correct errors arising from a concatenation of p DSD(1) channels, they can also correct errors arising from a concatenation of p DED(1) channels and thus a DED(p) channel, leading to the following theorem.

Theorem 93. *The codes in Constructions 85 and 90 can correct any number of duplications and at most p edit errors.*

3.5.6 The labeling function

In this subsection, we first present the labeling function f such that $f(\boldsymbol{x}) \neq f(\boldsymbol{y})$ for $\boldsymbol{y} \in B_{\mathrm{Irr}}^{\leqslant p}(\boldsymbol{x})$, used in Construction 85. By Theorem 82, $\boldsymbol{z} \in R(\mathcal{D}^{\leqslant p}(\boldsymbol{x})) \cap R(\mathcal{D}^{\leqslant p}(\boldsymbol{y}))$ can be obtained from $\boldsymbol{x}$ and from $\boldsymbol{y}$ by at most $2p\mathcal{L}$ indels. Hence, it suffices to find f such that $f(\boldsymbol{x}) \neq f(\boldsymbol{y})$ if there is a string $\boldsymbol{z}$ that can be obtained from both $\boldsymbol{x}$ and $\boldsymbol{y}$ through $2p\mathcal{L}$ indels. Note that since we are utilizing syndrome compression, choosing a more "powerful" labeling function does not increase the redundancy, which is still primarily controlled by $\max_{\boldsymbol{x}\in\mathrm{Irr}_q(n)} \|B_{\mathrm{Irr}}^{\leqslant p}(\boldsymbol{x})\|$. We use the next theorem for binary sequences to find f.

Theorem 94. *[74] There exists a labeling function $g : \{0,1\}^n \to \Sigma_{2^{\mathcal{R}(t,n)}}$ such that for any two distinct strings $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ confusable under at most t insertions, deletions, and substitutions, we have $g(\boldsymbol{c}_1) \neq g(\boldsymbol{c}_2)$, where $\mathcal{R}(t,n) = [(t^2+1)(2t^2+1) + 2t^2(t-1)] \log n + o(\log n)$.*

Since $\boldsymbol{z} \in R(\mathcal{D}^{\leqslant p}(\boldsymbol{x}))$ can be obtained from $\boldsymbol{x}$ via at most $2p\mathcal{L}$ indels, $\mathcal{U}_i(\boldsymbol{z})$ can be derived from $\mathcal{U}_i(\boldsymbol{x})$ by at most $2p\mathcal{L}$ indels, for $i \in [\lceil \log q \rceil]$. Based on Theorem 109 and the work [74], by letting $t = 2p\mathcal{L}$, we can obtain a labeling function g for recovering $\mathcal{U}_i(\boldsymbol{x})$ from $\mathcal{U}_i(\boldsymbol{z})$ under at most $2p\mathcal{L}$ indels. Therefore, $f : \Sigma_q^n \to \Sigma_{2^{\lceil \log q \rceil \mathcal{R}(t,n)}}$,

$$f(\boldsymbol{x}) = \sum_{i=1}^{\lceil \log q \rceil} 2^{\mathcal{R}(t,n)(i-1)} g(\mathcal{U}_i(\boldsymbol{x})),$$

where $t = 2p\mathcal{L}$, is a labeling function for the confusable sets $B_{\mathrm{Irr}}^{\leqslant p}(\boldsymbol{x})$, $\boldsymbol{x} \in \mathrm{Irr}_q(n)$. For each $\boldsymbol{x}$, a value $a_{\boldsymbol{x}}$ needs to be also determined such that $f(\boldsymbol{x}) \not\equiv f(\boldsymbol{y}) \bmod a_{\boldsymbol{x}}$ for $\boldsymbol{y} \in B_{\mathrm{Irr}}^{\leqslant p}(\boldsymbol{x})$. The existence of such $a_{\boldsymbol{x}}$, satisfying $\log a_{\boldsymbol{x}} \leqslant \log \|B_{\mathrm{Irr}}^{\leqslant p}(\boldsymbol{x})\| + o(\log n)$, is guaranteed by Theorem 102 provided that p is a constant (ensuring that $p^4 = o(\log \log n)$). The labeling function f and integers $a_{\boldsymbol{x}}$ are used in Construction 85.

In a similar manner, we can construct f' as a labeling function for $B_{\mathrm{Irr}}^{\leqslant p, \leqslant 2p\mathcal{L}}(\boldsymbol{x}), \boldsymbol{x} \in \mathrm{Irr}_q(n)$ and integers $a'_{\boldsymbol{x}}$, by setting $t = 4p\mathcal{L}$ to account for the deletion of length at most $2p\mathcal{L}$. This time, for all $\boldsymbol{x} \in \mathrm{Irr}_q(n)$, $\log a'_{\boldsymbol{x}} \leqslant \log \|B_{\mathrm{Irr}}^{\leqslant p, \leqslant 2p\mathcal{L}}(\boldsymbol{x})\| + o(\log n)$. The labeling function f' and integers $a'_{\boldsymbol{x}}$ are used in Construction 90.

3.5.7 The redundancy of the error-correcting codes

In this section, we study the rate and the redundancy of the codes proposed in Constructions 85 and 90 and compare these to those of the state-of-the-art short-duplication-correcting code given in [30], which has the highest possible asymptotic rate. For an alphabet of size q, the asymptotic rate of this code for short duplications is $\log \lambda$, where λ is the largest positive real root of $x^3 - (q-2)x^2 - (q-3)x - (q-2) = 0$ [8].

The following lemma shows that the code proposed in [30] essentially has size $\operatorname{Irr}_q(N)$, where N is the length of the code, a fact that will be helpful for comparing the redundancies of the codes proposed here with this baseline.

Lemma 95. *Let $\mathcal{C}_N^D$ be the code of length N over alphabet Σ_q introduced by [30] for correcting any number of duplication errors. For $q \geqslant 4$,*

$$\|\operatorname{Irr}_q(N)\| \leqslant \|\mathcal{C}_N^D\| \leqslant \frac{q-2}{q-3}\|\operatorname{Irr}_q(N)\|.$$

Proof. As shown in [30], $\|\mathcal{C}_N^D\| = \sum_{i=1}^{N} \|\operatorname{Irr}_q(i)\|$. Based on [83, Lemma 14], given $\boldsymbol{u} \in \operatorname{Irr}_q(N-1)$, there are at least $q-2$ choices for $a \in \Sigma_q$ such that $\boldsymbol{x} = \boldsymbol{u}a \in \operatorname{Irr}_q(N)$. Thus, $(q-2)\|\operatorname{Irr}_q(N-1)\| \leqslant \|\operatorname{Irr}_q(N)\|$ and, consequently, $\|\operatorname{Irr}_q(N-i)\| \leqslant \frac{\|\operatorname{Irr}_q(N)\|}{(q-2)^i}$. Then we have

$$\frac{\sum_{i=1}^{N} \|\operatorname{Irr}_q(i)\|}{\|\operatorname{Irr}_q(N)\|} \leqslant \sum_{j=0}^{N-1} \frac{1}{(q-2)^j} \leqslant \frac{q-2}{q-3}.$$

□

We now compare the redundancy of the code $\mathcal{C}^A$ of Construction 85 with the code $\mathcal{C}^D$ of [30] for correcting only duplications. The length N of $\mathcal{C}_n^A$ is $N = n + |\boldsymbol{r}|$, where

$$\begin{aligned} |\boldsymbol{r}| &= 2\log_q a_{\boldsymbol{x}} \\ &\leqslant 2\log_q \|B_{\operatorname{Irr}}^{\leqslant p}(\boldsymbol{x})\| + o(\log_q n) \\ &\leqslant 4p\log_q n + o(\log_q n) \end{aligned} \tag{3.13}$$

symbols. Hence, $N = n + 4p\log_q n + o(\log_q n)$. Then, the difference in redundancies between $\mathcal{C}_n^A$ and $\mathcal{C}_N^D$, both of length N, is

$$\begin{aligned} \log_q \|\mathcal{C}_N^D\| - \log_q \|\mathcal{C}_n^A\| &= \log_q \frac{\|\operatorname{Irr}_q(N)\|}{\|\operatorname{Irr}_q(n)\|} + O(1) \\ &\leqslant \log_q q^{N-n} + O(1) \\ &\leqslant 4p\log_q n + o(\log_q n), \end{aligned} \tag{3.14}$$

where the equality follows from Lemma 95 and the first inequality from the fact that $\|\operatorname{Irr}_q(i+1)\| \leqslant q\|\operatorname{Irr}_q(i)\|$. Noting that $\log_q n = \log_q N + o(\log_q N)$ yields the following theorem.

Theorem 96. *For constants $q \geqslant 4$ and p, the redundancy of the code $\mathcal{C}_n^A$ of length N is larger than the redundancy of the code $\mathcal{C}_N^D$ of the same length by at most $4\log_q N + o(\log_q N)$.*

We now turn our attention to comparing the redundancy of $\mathcal{C}_n^B$ of length N with $\mathcal{C}_N^D$. Here, $N - n = |\boldsymbol{r}| + O(1) = |\mathcal{E}_1(a'_{\boldsymbol{x}}, f'(\boldsymbol{x}) \bmod a'_{\boldsymbol{x}})| + O(1)$. Similar to (3.14), the extra redundancy is then $|\boldsymbol{r}| + O(1)$, which through $a'_{\boldsymbol{x}}$ depends on $\|B_{\operatorname{Irr}}^{\leqslant p, \leqslant 2p\mathcal{L}}(\boldsymbol{x})\|$, investigated in the next lemma.

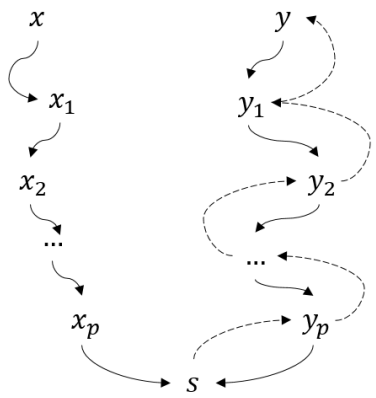


Figure 3.9: $\boldsymbol{s}$ results from passing $\boldsymbol{x}$ and $\boldsymbol{y}$ through a concatenation of p DSD(1) channels and a channel deleting a suffix of length at most $2p\mathcal{L}$ (c.f. Figure 3.7).

Lemma 97. *For $\boldsymbol{x} \in \mathrm{Irr}_q(n)$ with $q \geqslant 3$,*

$$\|B_{\mathrm{Irr}}^{\leqslant p, \leqslant 2p\mathcal{L}}(\boldsymbol{x})\| \leqslant q^{4p\mathcal{L}}(n + p\mathcal{L})^{2p}.$$

Proof. The proof is similar to that of Theorem 83, but also takes into account the effect of the suffix deletions, as shown in Figure 3.9. We have

$$\begin{aligned}\|B_{\mathrm{Irr}}^{\leqslant p, \leqslant 2p\mathcal{L}}(\boldsymbol{x})\| &\leqslant (968q(n + p\mathcal{L}) + 1)^{2p}(2p\mathcal{L} + 1)(2p\mathcal{L}q^{2p\mathcal{L}} + 1) \\ &\leqslant (2p\mathcal{L} + 1)^2 q^{2p\mathcal{L}}(968q + 1)^{2p}(n + p\mathcal{L})^{2p} \\ &\leqslant q^{4p\mathcal{L}}(n + p\mathcal{L})^{2p}.\end{aligned}$$

In the first line, $(968q(n+p\mathcal{L})+1)^{2p}$ is derived based on Theorem 83; $(2p\mathcal{L}+1)$ bounds the number of ways $\boldsymbol{s}$ can be obtained from $\boldsymbol{x}_p$ through a suffix deletion of length at most $2p\mathcal{L}$; and $(2p\mathcal{L}q^{2p\mathcal{L}}+1)$ bounds the number of ways $\boldsymbol{y}_p$ can be obtained from $\boldsymbol{s}$ by appending a sequence of length at most $2p\mathcal{L}$. The third line is obtained by noting that $(968q + 1)^{2p}(2p\mathcal{L} + 1)^2 \leqslant q^{2p\mathcal{L}}$ with $\mathcal{L} = 17$. □

Lemma 98. *For constants $q \geqslant 4$ and p, and $\boldsymbol{x} \in \mathrm{Irr}_q(n)$, the length $|\boldsymbol{r}|$ of $\boldsymbol{r} = \mathcal{E}_1(a'_{\boldsymbol{x}}, f'(\boldsymbol{x}) \bmod a'_{\boldsymbol{x}})$ satisfies*

$$|\boldsymbol{r}| \leqslant 8p \log_q n + o(\log_q n).$$

Proof. From the previous subsection, assuming p is a constant, we have that $\log a'_{\boldsymbol{x}} \leqslant \log \|B_{\mathrm{Irr}}^{\leqslant p, \leqslant 2p\mathcal{L}}(\boldsymbol{x})\| + o(\log n) \leqslant 2p \log n + o(\log n)$. Since $(f'(\boldsymbol{x}) \bmod a'_{\boldsymbol{x}}) \leqslant a'_{\boldsymbol{x}}$, we need $4p \log n + o(\log n)$ bits to represent the pair $(a'_{\boldsymbol{x}}, f'(\boldsymbol{x}) \bmod a'_{\boldsymbol{x}})$. Then, by Lemma 88, $|\mathcal{E}_1(a'_{\boldsymbol{x}}, f'(\boldsymbol{x}) \bmod a'_{\boldsymbol{x}}))| \leqslant 4p \log n(1 + o(1))/\log(q-2)$. The lemma follows from $\frac{\log q}{\log(q-2)} \leqslant 2$ for $q \geqslant 4$. □

Using Lemma 98, the next theorem gives the extra redundancy of correcting p substitutions compared to [30] and shows that there is no relative asymptotic rate penalty.

Theorem 99. *For constants $q \geqslant 4$ and p, the redundancy of the code $\mathcal{C}_n^B$ of length N is larger than the redundancy of the code $\mathcal{C}_N^D$ of the same length by at most $8\log_q N + o(\log_q N)$. The codes have the same asymptotic rate, which, for $q = 4$, equals $\log 2.6590$.*

3.5.8 Time complexity of encoding and decoding

Suppose $q \geqslant 4$ is a constant. The time complexities of both the encoding and decoding processes are polynomial in the lengths of the stored and retrieved sequences, respectively. The encoding process consists of four main parts:

1. Generating $\boldsymbol{x} \in \mathrm{Irr}_q(n)$ by the state-splitting algorithm, which has polynomial-time complexity [8].

2. Determining $\boldsymbol{b}_{\boldsymbol{x}}$ such that $\boldsymbol{x}\boldsymbol{b}_{\boldsymbol{x}}\boldsymbol{\sigma} \in \mathrm{Irr}_q(*)$, which has constant time complexity as the relevant subgraph of the De Bruijn graph (see Lemma 89) has a constant size (no more than q^5 vertices).

3. Determining $a'_{\boldsymbol{x}}$ and $f'(\boldsymbol{x}) \bmod a'_{\boldsymbol{x}}$. This is done in three steps, with polynomial time complexity. i) Given $\boldsymbol{x} \in \mathrm{Irr}_q(n)$, we find the elements of a set $\hat{B} \supseteq B_{\mathrm{Irr}}^{\leqslant p, \leqslant 2p\mathcal{L}}(\boldsymbol{x})$ whose size satisfies the upper bound given in Lemma 97. Specifically, given $\boldsymbol{x}$ we find all sequences that can be obtained from it through $\leqslant p$ *short substring substitutions*, one deletion of a suffix of length $\leqslant 2p\mathcal{L}$, one insertion of a suffix of length $\leqslant 2p\mathcal{L}$, and another $\leqslant p$ short substring substitutions, where in each short substring substitution step, we replace a substring $abcde \in \mathrm{Irr}_q(5)$ by another irreducible substring from the set $R(\mathcal{D}^1(abcde))$ and then deduplicate all copies. The total time complexity of this step is $O(n^{2p})$ as each element of $\hat{B}$ is obtained by a bounded number of operations and $\|\hat{B}\| = O(n^{2p})$. ii) Since computing $f'(\cdot)$ from [74] has time complexity $O(n \log n)$, computing it for all elements of $\hat{B}$ takes $O(n^{3p} \log n)$ steps. iii) Computing the remainder of these values modulo the $\leqslant 2^{\log O(n^{2p})}$ possible values for $a'_{\boldsymbol{x}}$ also has polynomial complexity.

4. Generating $\boldsymbol{r} = \mathcal{E}_1(a'_{\boldsymbol{x}}, f'(\boldsymbol{x}) \bmod a'_{\boldsymbol{x}})$ using the encoder $\mathcal{E}_1$ for the code in Construction 68, which has complexity polynomial in $|\boldsymbol{r}|$ based on Subsection 3.4.4. Hence, by Lemma 98, the complexity is at most polynomial.

Therefore, when p is a constant, the time complexity of the encoding process is polynomial with respect to N (as well as n).

Decoding requires finding the root of the retrieved word, which is linear in its length; decoding $a'_{\boldsymbol{x}}$ and $f'(\boldsymbol{x}) \bmod a'_{\boldsymbol{x}}$, which is polynomial as discussed in Subsection 3.4.4; and determining $\boldsymbol{x}$ through a brute-force search among all inputs that can lead to the same $(n - p\mathcal{L})$-prefix of the root of the retrieved sequence. Similar to the discussion about finding $\hat{B}$ above, the brute-force search is polynomial in n. Hence, decoding is polynomial in the length of the retrieved sequence.

3.6 Summary

This chapter focused on correcting any short duplications and at most p edit errors by three steps: i) correcting short duplications and at most one edit, ii) correcting short duplications and at most p substitutions, iii) correcting short duplications and at most p edits with low redundancy.

This chapter first considered constructing error-correcting codes for channels with many short duplications and one edit error. Because the channel allows an arbitrary number of duplications, a single edit may affect an unbounded segment of the output. For example, an inserted symbol may appear many times in different positions. However, with an appropriate construction of message blocks and processing of the output strings, the edit error leads to the erasure of at most 4 message blocks or substitution of at most 2. Therefore, a maximum distance separable (MDS) code in Construction 52 with minimum Hamming distance 5 over message blocks can correct these errors. When $q = 4$, the case corresponding to DNA storage, a computational bound for the code rate shows that the asymptotic rate is only 0.003 bits/symbol smaller than that of the code that corrects short duplications but no edits.

Motivated by the work for an extra edit error, this chapter next extended the previous work to correct short duplications and at most p substitutions. Based on the preprocessing for one edit error, the effect of short duplications and at most p substitutions can be transformed as at most p substring edit errors over constrained sequences. Different from one substring edit that causes erasure or substitution errors, multiple substring edits poses the challenge of identifying error-free message blocks. By concatenating at least $3p$ blocks with different colors as a message group, we may detect at most $2p$ substitutions or erasures of message groups in the decoder. Hence, we proposed an extended MDS code in Construction 68 to correct another at most p substitutions, with a loss of asymptotic code rate. Note that the construction is applied as an auxiliary code to protect the syndrome information in the following.

In order to further reduce the rate loss, we introduced codes for correcting any number of duplication and at most p edit errors simultaneously. Recall that the set of irreducible strings is a code capable of correcting short duplication errors with low redundancy. To additionally correct edit errors, we append to each irreducible sequence $\boldsymbol{x}$ of length n a vector generated through syndrome compression that enables us to distinguish confusable inputs. Given that edit and duplication errors manifest as substring edit errors, we designed a buffer and the auxiliary code in a way to enable us to recover the syndrome information from the received string. In each step of the construction, we carefully ensured that the resulting sequence is still irreducible. The additional redundancy compared to the codes correcting duplications only [30] is $8p(\log_q n)(1 + o(1))$, with the number of edits p and the alphabet size q being constants and $q \geqslant 4$. This additional redundancy is at most a factor of 2 away from the lowest-redundancy codes for correcting p edits only [73] and a factor of 4 away from the GV bound given in Theorem 84. The encoding and decoding processes have polynomial time complexities. We focused on $q \geqslant 4$ as it includes the case with the most practical importance, i.e., $q = 4$. While not all the results of the chapter are valid for $q = 3$ (e.g., the bound on L' in Lemma 88), we expect many of the ideas to be applicable to this case.

Chapter 4

Correcting a substring edit with bounded length

4.1 Introduction

As discussed in Section 1.2.2, localized errors including burst substitutions/insertions/deletions or localized deletions [2], [9], [17], [24], [42], [65], [66], [91], [93], [104] are widely observed in various applications such as wireless communication, disk data storage, DNA storage, and document synchronization. For example, the duplications discussed in previous chapters can all be viewed as localized errors. The current chapter focuses on correcting a *k-substring edit* error with as low redundancy as possible, which replaces one substring with another string at the same position, both with lengths bounded by k. Note that one motivation is that burst deletions, burst insertions, burst substitutions, and deletions/insertions occurring in a bounded window can all be considered as special cases of substring edits with a bounded length.

Unlike prior works that assume the prevalence of burst errors, we first present an experimental analysis to statistically validate this hypothesis. Based on two datasets, two versions of the source code of the bash shell and the DNA sequencing data exposed to errors, statistical tests provide strong evidence against the null hypothesis, namely that errors/edits are distributed uniformly in the sequences. Moreover, we study the suitability of substring-edit-correcting codes for error correction and data synchronization. For each data sequence and its edited version, the experiment shows that substring-edit-correcting codes achieve lower redundancy than general indel/substitution-correcting codes.

Our problem is related to the problem of deletion-correcting codes and burst-deletion-correcting codes. However, as we will show in Lemma 100, a code that can correct a burst of at most ℓ deletions (or one that can correct a burst of at most ℓ insertions) cannot necessarily correct a k-substring edit, even if ℓ is much larger than k. On the other hand, a k-substring edit can be corrected by a code that can correct 2 bursts of at most k deletions or, as we show in Lemma 100, by a code that can correct at most $2k$ deletions. These observations lead to the conclusion that existing codes for correcting multiple deletions [72], [74] or multiple bursts of deletions [86] can correct a k-substring

edit with redundancy at least $4k \log n$ [74]. The goal of the current work is to construct codes that can correct a k-substring edit with less redundancy.

While codes for correcting a burst of at most k deletions cannot correct a k-substring edits, the idea of first identifying an approximate location for the error presented in Lenz et al. [42] and Bitar et al. [2] using the position of specific patterns is useful for our problem. We divide k-substring edits into *strict* k-substring edits (that will change the length in the outputs) and bursts of at most k substitutions, referred to as k-burst substitutions. For strict edits, we first extend the codes in [2], [42] to locate the error to be in an interval of length $O((\log n)^2)$ and then correct it. For k-burst substitutions, which cannot be located using the patterns mentioned above, we adapt the Fire code [17]. Then we combine the two error-correcting codes in a manner that enables polynomial-time encoding and decoding.

The chapter is organized as follows. Section 4.2 presents the notation and preliminaries. Section 4.3 discusses the prevalence of burst errors in real data and the utility of substring-edit-correcting codes. Finally, Section 4.4 presents the code constructions and an analysis of the time complexity of encoding/decoding and the redundancy.

4.2 Notation and preliminaries

4.2.1 Notation

Without loss of generality, let $\Sigma_q = \{0, 1, \ldots, q-1\}$ be a finite alphabet of size q. The set of length-n strings and finite strings over Σ_q are denoted as Σ_q^n and Σ_q^*, respectively. The empty string, denoted Λ, is also considered a member of Σ_q^*. In this chapter, we focus on the binary alphabet Σ_2. For $a, b \in \mathbb{Z}$, let $[a, b] = \{a, a+1, \ldots, b\}$ and $[b] = [1, b]$. Unless otherwise stated, logarithms are to the base of 2.

For $\boldsymbol{x}, \boldsymbol{y} \in \Sigma_2^n$, let $\boldsymbol{x}_{[a,b]} = x_a x_{a+1} \cdots x_b$ and let $\boldsymbol{xy}$ and $(\boldsymbol{x}, \boldsymbol{y})$ denote the concatenation of $\boldsymbol{x}, \boldsymbol{y}$. For $\boldsymbol{x}, \boldsymbol{v} \in \Sigma_q^*$, $\boldsymbol{v}$ is a substring of $\boldsymbol{x}$ if $\boldsymbol{x} = \boldsymbol{uvw}$ for some $\boldsymbol{u}, \boldsymbol{w} \in \Sigma_q^*$. Furthermore, $|\boldsymbol{x}|$ is the length of a sequence $\boldsymbol{x}$ and $\|S\|$ is the number of elements in a set S. Given an integer r and a symbol $a \in \Sigma_2$, let a^r denote a *run* of r consecutive symbols a.

4.2.2 The k-substring edit channel

Given a string $\boldsymbol{x}$, a *k-burst deletion* (resp. *insertion*) in $\boldsymbol{x}$ removes (resp. inserts) at most k consecutive symbols, while a *k-substring edit* replaces a substring $\boldsymbol{v}$ of $\boldsymbol{x}$ by another string $\boldsymbol{v'}$, where $|\boldsymbol{v}|, |\boldsymbol{v'}| \leqslant k$ and at least one of $\boldsymbol{v}, \boldsymbol{v'}$ is non-empty. The k-substring edit is a *k-burst substitution* if $|\boldsymbol{v}| = |\boldsymbol{v'}|$ and a *strict* k-substring edit otherwise. In particular, $\boldsymbol{v'} = \Lambda$ results in a burst deletion addressed by previous works. For example, given $\boldsymbol{x} = 1001\underline{11}01\underline{1}101 \in \Sigma_2^n$, a 4-substring edit may generate $\boldsymbol{z} = 1001\underline{0}10\underline{1}101$ by replacing $\boldsymbol{x}_{[5,8]} = 1101$ with $\boldsymbol{z}_{[5,7]} = 010$.

The next lemma discusses the relationship between deletion-correcting codes and codes that can correct a k-substring edit.

Lemma 100. *The codes in the statements below are over Σ_q^n, where $n, q \geqslant 2$. Let k be a positive integer.*

1. *A code that can correct 2 k-burst deletions can correct a k-substring edit.*
2. *For any $\ell < n$, there exists a code that can correct a burst of at most ℓ deletions but not a k-substring edit.*
3. *For any $\ell < n$, if $\ell < 2k$, then there exists a code that can correct up to ℓ deletions but not a k-substring edit.*

Proof. 1. For two words $\boldsymbol{x}, \boldsymbol{y}$, if $\boldsymbol{z}$ can be obtained from each by a k-substring edit, then there exists a $\boldsymbol{z}'$ that can be obtained from each by 2 k-burst deletions, by also deleting the inserted strings from $\boldsymbol{z}$.

2. Consider the code $\mathcal{C} = \{0^n, 10^{n-2}1\}$, which can correct any burst of at most $\ell < n$ deletions since 0^n can only produce $0^m, m \in [n]$, while the $10^{n-2}1$ cannot. Note that $0^{n-1}1$ can be obtained from both 0^n and $10^{n-2}1$ through a single substitution. So $\mathcal{C}$ cannot correct a k-substring edit for $k > 0$.

3. Let $h = \min\{n, 2k\}$ and note that $\ell < h$. Consider the code $\{0^h\boldsymbol{v}, 1^h\boldsymbol{v}\}$, $\boldsymbol{v} \in \Sigma_q^{n-h}$, which can correct any $\leqslant \ell$ deletions. However, each of the codewords can produce $0^{\lfloor h/2 \rfloor}1^{\lceil h/2 \rceil}\boldsymbol{v}$ via a k-substring edit.

□

Given a code $\mathcal{C} \subseteq \Sigma_q^n$, the redundancy of the code $\mathcal{C}$ is defined as $n \log q - \log \|\mathcal{C}\|$. For binary, which is the focus of our code construction, part 1 of the above lemma implies the code given in [74] for correcting $\leqslant 2k$ deletions can correct a k-substring edit over the binary alphabet with the redundancy of roughly $8k \log n$ bits. If the k-substring edit is viewed as at most k insertions, deletions, or substitutions, the code given in [74] requires redundancy of roughly $4k \log n$ bits. To the best of our knowledge, that is the lowest redundancy that can be achieved by an existing code for this problem. The code we present in Theorem 118 has redundancy of roughly $2 \log n$.

Given a string $\boldsymbol{x} \in \Sigma_q^n$, let $D_{b,k}(\boldsymbol{x}) \subseteq \Sigma_q^*$ denote the set of strings generated from $\boldsymbol{x}$ by at most b k-substring edit errors and let $B_{b,k}(\boldsymbol{x}) \subseteq \Sigma_q^n$ denote the *confusable set* of $\boldsymbol{x}$, i.e., the set of sequences $\boldsymbol{y}$ other than $\boldsymbol{x}$ for which $D_{b,k}(\boldsymbol{x}) \cap D_{b,k}(\boldsymbol{y}) \neq \varnothing$. When $b = 1$ and k is clear from the context, we use $D(\boldsymbol{x}), B(\boldsymbol{x})$ instead of $D_{b,k}(\boldsymbol{x}), B_{b,k}(\boldsymbol{x})$.

We now find the *Gilbert-Varshamov (GV) bound* on the size of the code. Define

$$r_n(b,k) = \max_{\boldsymbol{x} \in \Sigma_q^n} \|B_{b,k}(\boldsymbol{x})\| + 1.$$

Lemma 101. *Assuming an alphabet of size q, we have*

$$r_n(b,k) \leqslant (n + bk)^{2b}(k+1)^{4b}q^{2kb}$$

and there exists a code $\mathcal{C} \subseteq \Sigma_q^n$ of length n capable of correcting at most b k-substring edits with the size at least

$$\|\mathcal{C}\| \geqslant \frac{q^n}{r_n(b,k)}.$$

Proof. For $\boldsymbol{x} \in \Sigma_q^n$, let $B_{b,k}(\boldsymbol{x}) \subseteq \Sigma_q^n$ denote the set of sequences that can produce the same output as $\boldsymbol{x}$ by at most b k-substring edits. That is $\boldsymbol{y} \in B_{b,k}(\boldsymbol{x})$ if and only if there exists $\boldsymbol{z}$ that can be produced from both $\boldsymbol{x}$ and $\boldsymbol{y}$ through at most b k-substring edit errors. Furthermore, each k-substring edit is reversible, i.e., if $\boldsymbol{x}_i \in D(\boldsymbol{x}_{i-1})$, then $\boldsymbol{x}_{i-1} \in D(\boldsymbol{x}_i)$. Then each $\boldsymbol{y} \in B_{b,k}(\boldsymbol{x})$ can be generated from $\boldsymbol{x}$ by $\boldsymbol{x} \to \boldsymbol{x}_1 \to \cdots \to \boldsymbol{x}_{b-1} \to \boldsymbol{z} \to \boldsymbol{y}_{b-1} \to \cdots \to \boldsymbol{y}_1 \to \boldsymbol{y}$ by $2b$ k-substring edits, where $\boldsymbol{x} = \boldsymbol{x}_0$ and $\boldsymbol{y} = \boldsymbol{y}_0$. This sequence of edits consists of at most $2b$ burst deletions and 2b burst insertions. There are $\leqslant (k+1)^{4b}$ ways to choose their lengths and $\leqslant q^{2kb}$ to choose the inserted strings. We claim, to be proved later, that for each string, there are at most $n+bk$ possible positions for the edit, yielding the Lemma.

To prove the claim, note that for a string of length m, there are $m+1$ possible positions for a substring edit that involves only an insertion and m positions if the edit contains a nontrivial deletion. The strings $\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{z}$ either have length less than $n+bk$ or if their length is equal to $n+bk$ (only possible for $\boldsymbol{z}$), then the edit must contain a deletion. □

Assuming b, k are constants, the redundancy is bounded above by $2b \log n + o(\log n)$, which is the same as the redundancy of the codes proposed in this chapter for $b = 1$ and the binary alphabet.

4.2.3 Relevant prior results

We first recall a result from *syndrome compression*, a technique used to construct codes with low redundancy [75], to our problem.

Theorem 102 (c.f. [75, Theorem 5])**.** *Given $\boldsymbol{x} \in \Sigma_2^n$, let $f : \Sigma_2^n \to \Sigma_{2^{\mathcal{R}(n)}}$ be a (labeling) function over the confusable set $B(\boldsymbol{x})$ such that $f(\boldsymbol{x}) \neq f(\boldsymbol{y})$ for every $\boldsymbol{y} \in B(\boldsymbol{x})$, where $\mathcal{R}(n) = o(\log\log n \cdot \log n)$. Then there exists an integer $a \leqslant 2^{\log \|B(\boldsymbol{x})\| + o(\log n)}$ such that for all $\boldsymbol{y} \in B(\boldsymbol{x})$, we have $f(\boldsymbol{x}) \not\equiv f(\boldsymbol{y}) \bmod a$.*

We will use the following definitions and results from [2], [42]. These works correct a burst of deletions with low redundancy by first locating the approximate position of the error.

Given sequences $\boldsymbol{x} \in \Sigma_2^n$ and a *pattern* (string) $\mathcal{P}$, define $\mathbf{1}_{\mathcal{P}}(\boldsymbol{x}) \in \Sigma_2^n$ as the indicator vector whose ith element is 1 if $\boldsymbol{x}_{[i,i+|\mathcal{P}|-1]} = \mathcal{P}$ and is 0 if $\boldsymbol{x}_{[i,i+|\mathcal{P}|-1]} \neq \mathcal{P}$ or $i + |\mathcal{P}| - 1 > n$. Further, let $n_{\mathcal{P}}(\boldsymbol{x})$ denote the number of 1's in $\mathbf{1}_{\mathcal{P}}(\boldsymbol{x})$ and $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})$ represent a length-$(n_{\mathcal{P}}(\boldsymbol{x})+1)$ vector whose jth element is the distance between positions of the $(j-1)$-th and the jth 1 in the string $(1, \mathbf{1}_{\mathcal{P}}(\boldsymbol{x}), 1)$ for $j \in [n_{\mathcal{P}}(\boldsymbol{x})+1]$. A sequence $\boldsymbol{x}$ is $(\mathcal{P}, \delta)$-dense if each interval of length δ in $\boldsymbol{x}$ contains at least one substring $\mathcal{P}$. The set of $(\mathcal{P}, \delta)$-dense binary strings of length n is denoted $\mathcal{D}_{\mathcal{P},\delta}(n)$. Based on [42, Lemma 1], for $\mathcal{P} = 0^k 1^k$, $n \geqslant 5$, and $\delta = k2^{2k+1}\lceil \log n \rceil$, we have

$$|\mathcal{D}_{\mathcal{P},\delta}(n) \cap \Sigma_2^n| \geqslant 2^{n-1}. \tag{4.1}$$

Furthermore, for $j \geqslant 2$, we have $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})_j \geqslant 2k$ due to the length of $\mathcal{P}$.

Given an integer string $\boldsymbol{x} \in Z^n$, define the *Varshamov-Tenengolts (VT) check sum* as $VT(\boldsymbol{x}) = \sum_{i=1}^{n} ix_i$. We next recall the code in [2] used to locate the burst of deletions or localized deletions in an interval.

Lemma 103 (cf. [2]). *Given integers $c_1 \in [0, 4]$, $c_2 \in [0, 6n-1]$, and $\delta = k2^{2k+1}\lceil \log n \rceil$, there exists a code*

$$\mathcal{C}_d = \{\boldsymbol{x} \in \Sigma_2^n \cap \mathcal{D}_{\mathcal{P},\delta}(n), n_{\mathcal{P}}(\boldsymbol{x}) = c_1 \mod 5, \\ VT(\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})) = c_2 \mod 6n.\}$$

that can locate a burst of deletions or localized deletions in an interval with length $O((\log n)^2)$.

The key notations used in the chapter are summarized in Table 4.1.

Table 4.1: Key notations in Chapter 4

Notation	Definition
$\Sigma_q = \{0, 1, \cdots, q-1\}$	the alphabet set with q elements
$[a, b]$	the set of consecutive integers $\{a, a+1, \cdots, b\}$
$\|\|S\|\|$	the number of elements in the set S
k-substring edit	generate $\boldsymbol{y} = \boldsymbol{uv'w}$ from $\boldsymbol{x} = \boldsymbol{uvw}$ by replacing the substring $\boldsymbol{v}$ with another string $\boldsymbol{v'}$ with $\|\boldsymbol{v}\|, \|\boldsymbol{v'}\| \leqslant k$
$D_{1,k}(\boldsymbol{x})/D(\boldsymbol{x})$	the set of all outputs generated from $\boldsymbol{x}$ by a k-substring edit
$\mathcal{P}$	the pattern (string), i.e., 0^k1^k
$n_{\mathcal{P}}(\boldsymbol{x})$	the number of patterns in $\boldsymbol{x}$
$\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})$	the length-$(n_{\mathcal{P}}(\boldsymbol{x})+1)$ vector representing the distance between two adjacent patterns in $\boldsymbol{x}$
$(\mathcal{P}, \delta)$-dense string	each interval of length-δ in $\boldsymbol{x}$ contains at least one pattern $\mathcal{P}$
$\mathcal{D}_{\mathcal{P},\delta}(n)$	the set of all $(\mathcal{P}, \delta)$-dense strings
$VT(\boldsymbol{x})$	$\sum_{i=1}^{n} ix_i$

4.3 Substring edits in nanopore sequencing and document editing

In this section, we investigate the hypothesis that in real-world settings, errors/edits commonly occur in a bursty manner, rather than being distributed uniformly. We also study whether substring-edit-correcting codes can achieve lower redundancy than general edit-correcting codes. We performed experimental studies on two real-world datasets, corresponding to two applications of the codes:

- *Error-correction:* We investigate the set of errors encountered in nanopore sequencing [12] in DNA data storage. The data consists of 1000 input-output pairs, where the input represents the (true) base sequence of a DNA molecule. Each input sequence is of length 200, with bases randomly generated from $\{A, C, G, T\}$. The output represents the sequence detected by

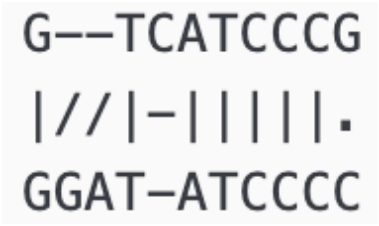


Figure 4.1: An alignment of two DNA sequences, where the top sequence can be obtained from the bottom one via deletions (/), insertions (−), and substitutions (·).

nanopore, as simulated by using the nanopore deep simulator [47] and the Nanopore's Guppy basecaller.

- *Data synchronization:* Suppose Alice, who knows only $\boldsymbol{x}$, needs to communicate $\boldsymbol{x}$ to Bob, who knows only $\boldsymbol{z}$. Consider two documents $\boldsymbol{x}$ and $\boldsymbol{z}$, where $\boldsymbol{x}$ is the edited version of $\boldsymbol{z}$. This task is referred to as *data synchronization*, and can be accomplished using error-correcting codes [60]. We study the characteristics of the editing process, which affects the number of bits needed for synchronization. The dataset consists of versions 5.0 and 5.1 of the source code for the Bash utility[1], containing 1301 and 1383 files, respectively, where each common file in version 5.1 is viewed as an edited version of the one in version 5.0.

Recall that our goal is to determine whether edits/errors are i) bursty or ii) uniformly/independently distributed. To rigorously answer this question, one way is to first define a uniform/independent random process for errors and edits and then use hypothesis testing to determine if such a model explains the observed data, which will be discussed in Subsection 4.3.2. First, however, we perform an intuitive but less rigorous test over the alignment of pairs of sequences in Subsection 4.3.1. Finally, in Subsection 4.3.2, we consider choosing the model that leads to the lowest cost in error-correction and synchronization tasks for our data.

4.3.1 Independence test on alignment

Let $\boldsymbol{z}$ be the erroneous/edited version of $\boldsymbol{x}$. An alignment between $\boldsymbol{x}$ and $\boldsymbol{z}$ identifies the positions where the sequences match and how they differ (see Figure 4.1).

From the alignment, let us produce the binary sequence, denoted $\boldsymbol{a}$, in which 1 represents a match and 0 represents an edit (substitution, insertion, deletion). If edits/errors are distributed in a uniform manner over $\boldsymbol{x}$, e.g., resulting from an "independent" process, then it is reasonable to expect the alignment $\boldsymbol{a}$ to resemble an iid sequence. Therefore, we apply the Wald-Wolfowitz runs-test [51].

Wald-Wolfowitz runs-test

Consider a binary sequence $\boldsymbol{a} = a_0 a_1 \cdots a_{N-1}$. The Wald-Wolfowitz runs-test [51] tests the null hypothesis that $\boldsymbol{a}$ is iid generated. The number of runs R is used as the test statistic. Conditioned

[1] https://ftp.gnu.org/gnu/bash/

on the number of 1's t_1 and the number of 0's t_0, it can be shown that

$$\Pr(R = 2s|t_1, t_0) = \frac{2\binom{t_1-1}{s-1}\binom{t_0-1}{s-1}}{\binom{N}{t_1}}, \tag{4.2}$$

$$\Pr(R = 2s+1|t_1, t_0) = \frac{\binom{t_1-1}{s-1}\binom{t_0-1}{s} + \binom{t_1-1}{s}\binom{t_0-1}{s-1}}{\binom{N}{t_1}}. \tag{4.3}$$

where there are $\binom{N-1}{s-1}$ choices to put N balls into s bins each with at least one ball. Note that either too large or too small values of R imply dependence between a_i's, thus suggest to reject the null hypothesis. However, in all examples we tested, the actual number of runs r is small. Therefore, for simplicity, we consider the one-sided test with p-value $\Pr(R \leqslant r|t_1, t_0) = \sum_{y=1}^{r} \Pr(R = y|t_1, t_0)$.

Here, the null hypothesis is that $\boldsymbol{a}$ is iid generated and the number R of runs in $\boldsymbol{a}$ is the test statistic. Conditioned on the number of 0's t_0 and the number of 1's t_1, the p-value is $\Pr(R \leqslant r|t_0, t_1)$, as we do not expect to see very few runs in an iid sequence. As in "RUNS-TEST" column of Table 4.2, this test strongly suggests that edits are not iid for both datasets.

4.3.2 A probabilistic edit process

Another approach is to perform hypothesis testing on a probabilistic edit process. Again, let $\boldsymbol{x}$ be our data sequence of length n, and $\boldsymbol{z}$ its edited version. Based on an intuitive interpretation of "uniform edits", we define the following simple edit process: i) A random number K_i of insertions are uniformly distributed over the $n+1$ bins between x_i and x_{i+1}, $i = 0, \ldots, n$, where x_0 and x_{n+1} are defined as the empty symbol. ii) A random number K_d of substitutions and deletions are uniformly applied on $x_1, \ldots, x_n$.

The null hypothesis is that $\boldsymbol{z}$ is generated by this edit process. Since insertions occur independently of substitutions and deletions, we apply two separate tests. For insertions, we consider W, the number of non-empty bins as the test statistic. According to the proposed edit process, how are insertions located can be described by the problem of distributing s balls uniformly over the N bins. The probability of observing w non-empty bins can be shown as

$$\Pr(W = w|s, N) = \frac{\binom{N}{w} \cdot \binom{s-1}{w-1}}{\binom{N+s-1}{s}}, \tag{4.4}$$

where $\binom{N+s-1}{s}$ denotes all the cases of distributing s balls into N bins, and $\binom{N}{w} \cdot \binom{s-1}{w-1}$ denotes all the cases that w bins are not empty. If most insertions cluster in a small number of bins, i.e., W being small, then we reject the null hypothesis. The p-value $\Pr(W \leqslant w|s, N)$ is summarized in "INS-TEST" column of Table 4.2.

For substitutions and deletions, we consider again R, the number of runs in the edit pattern (excluding insertions) as the test statistic, and reject the null hypothesis if R is small. The results are given in "SUBDEL-TEST" column of Table 4.2. High rejection rates for both tests suggest that edits are not uniform.

subsectionOperational evaluation of error/edit models Given two sequences $\boldsymbol{x} \in \Sigma_q^n$ and $\boldsymbol{z} \in$

Σ_q^*, their differences can be described via b k-substring edits for a range of possible pairs (b, k). Operationally, the best description, i.e., (b, k) pair, is the one that leads to the lowest cost for the task at hand. For error-correction, where $\boldsymbol{z}$ is an erroneous copy of $\boldsymbol{x}$, the cost is the redundancy of the code that allows correcting the errors in $\boldsymbol{z}$. If $\boldsymbol{z}$ can be obtained from $\boldsymbol{x}$ via b k-substring edits, based on the GV bound, there exists such a code of length n with redundancy $\log r_n(b, k)$. For synchronization, where $\boldsymbol{x}$ is the edited version of $\boldsymbol{z}$ (or vice versa), the cost is the information exchange, i.e., the number of bits needed to be transmitted. It can be shown [60] that exchanging $\log r_n(b, k)$ bits is sufficient, achievable using a systematic code with n information symbols and $\log_q r_n(b, k)$ check symbols.

4.3.3 Experiment results

Our experiment starts with computing the alignment of data sequence pairs in both datasets. We consider the 1000 input-output pairs in the nanopore sequencing dataset and the 589 pairs of common files (with lengths at most 20000 bytes and contain edits) in the bash dataset. We use the conventional dynamic programming approach [55] for computing the alignment. Table 4.2 presents the fraction of sequences rejecting the null hypothesis at p-value threshold of 5% for two datasets for three thresholds.

	RUNS-TEST	INS-TEST	SUBDEL-TEST
Bash	97.2%	100%	97.6%
DNA	95.7%	94.6%	76.1%

Table 4.2: Fraction of sequences rejecting the null hypothesis at p-value threshold of 5%.

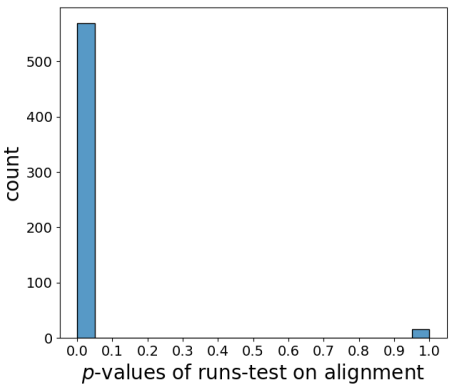
a) Runs-test on alignment

For each alignment, the runs-test is applied for determining if it resembles an iid sequence. In particular, the alignment is first converted into a binary edit pattern sequence with 1 representing a match and 0 one of the three types of edits. The p-value of the runs-test is computed according to (4.2) and (4.3), where t_1, t_0, r are the numbers of 1's, 0's, and runs in the edit pattern, respectively.

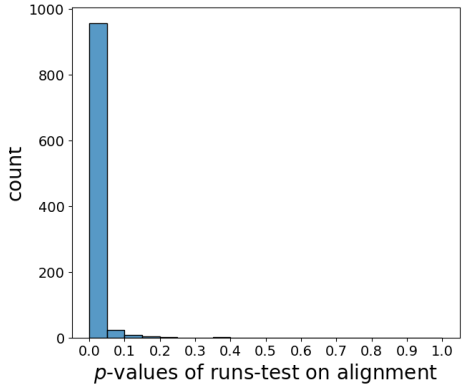
In our experiment, the runs-test is applied on all 1000 input-output pairs in the nanopore sequencing dataset and 589 pairs of edited files in the bash dataset. Table 4.2 contains fraction of sequences rejecting the null hypothesis at p-value threshold of 5%. We also provide histograms of all obtained p-values in Figure 4.2. It is clear that most alignment data does not suggest edits being uniform.

b) Testing the edit process

The second part of our analysis focuses on the two testings for the proposed edit process. We assume that the alignment between two data sequences gives the actual edits that happened. Let n be the length of the unedited data sequence.

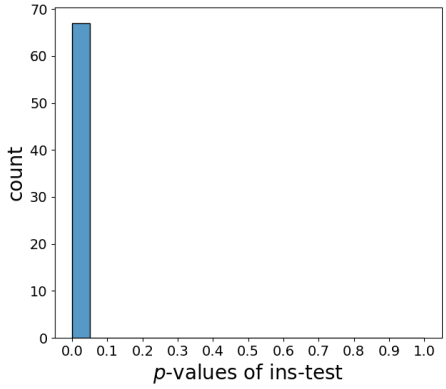


(i) Bash dataset

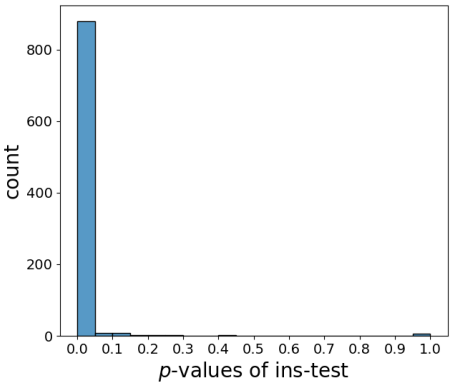


(ii) Nanopore sequencing dataset

Figure 4.2: Histograms of p-values for the runs-test applied on alignment.



(i) Bash dataset



(ii) Nanopore sequencing dataset

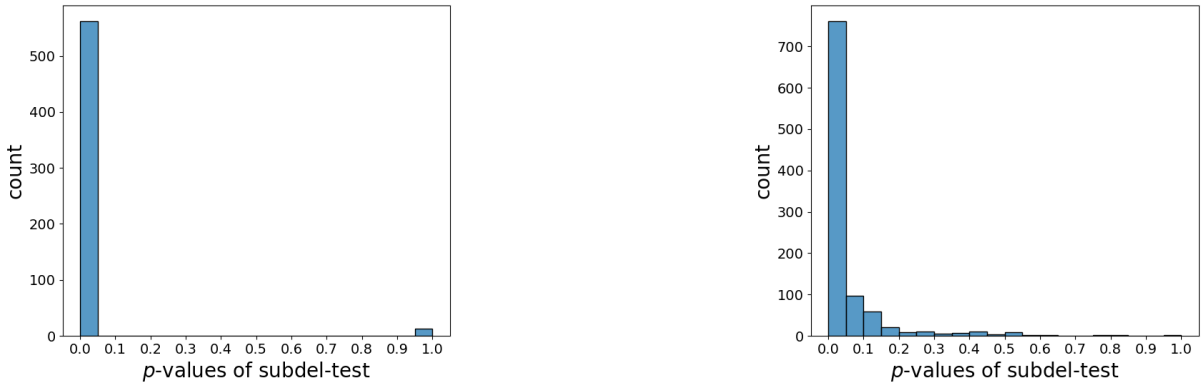
Figure 4.3: Histograms of p-values for INS-TEST.

- INS-TEST: For the insertions, we obtain the number of insertions K_i and the number of runs of insertions w from the alignment. The p-value is defined as the probability of seeing at most w non-empty bins, i.e., $\Pr(W \leqslant w | s = K_i, N = n + 1)$, where the pmf of W is given by (4.4).

- SUBDEL-TEST: For the substitutions and deletions, we first remove the insertions from the alignment. Next, we convert the alignment to the binary edit pattern and count the number of 1's K_d (corresponding to edits) and the number of 0's $n - K_d$. The test statistic is again R, the number of runs in the edit pattern. It is clear that under the assumed edit process, the distribution of R given K_d is given by (4.2) and (4.3), with $t_1 = K_d, t_0 = n - K_d$. The p-value is computed in the same way.

INS-TEST and SUBDEL-TEST are both applied on the two datasets, with fractions of rejections in Table 4.2. Note that we only apply INS-TEST on data sequences that contain at least 5 insertions, and only apply SUBDEL-TEST on data sequences that contain at least 1 deletion or substitution. Figures 4.3 and 4.4 show histograms of the p-values. Strong evidence for rejecting the uniform edit

process can be observed.



(i) Bash dataset (ii) Nanopore sequencing dataset

Figure 4.4: Histograms of p-values for SUBDEL-TEST.

c) Operational evaluation of error/edit models

For each pair of sequences in the genome data set, among all valid (b, k) pairs, we find the one that minimizes $r_n(b, k)$, where $n = 200, q = 4$. The histogram of the best (b, k) pairs is given in Figure 4.5, which indicates that viewing the errors as 13 2-substring edits minimizes the redundancy for the largest number of input-output pairs.[2] This suggests that edits with $k > 1$ better describe our data and codes correcting b k-substring edits for $k > 1$ are of use in DNA data storage. Note, however, that a priori we do not know the error that will occur and may have to over-provision to achieve reliability.

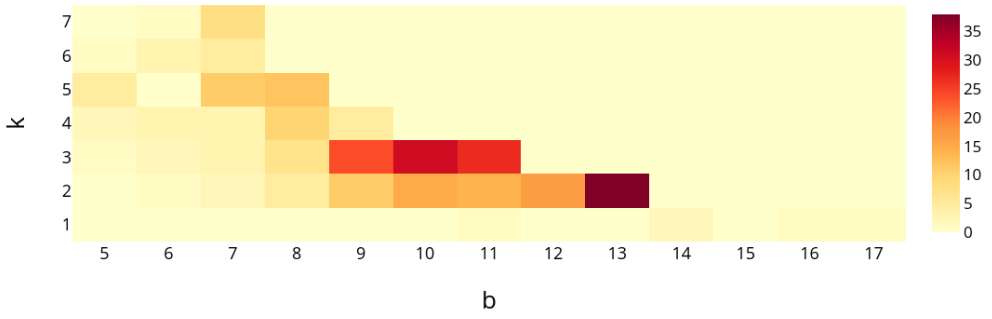


Figure 4.5: Histogram of optimal (b, k) values for the Nanopore sequencing dataset.

Similarly, for each edited file in the bash dataset, we find the (b, k) pair with the minimum redundancy, i.e., $\log r_n(b, k)$ with n being the unedited file size and alphabet size $q = 256$, among all valid pairs. In our experiment, we let k range from 1 to 10 and only consider files of lengths

[2]We point out that only 272 sequence pairs are included as the rest are so erroneous that they have their minimum redundancy $\log r_n(b, k)$ larger than the original length (400 bits for $n = 200, q = 4$).

smaller than 3000 bytes (392 in total) to avoid large running time. The valid (b, k) pairs are found by computing the smallest b for each k using a dynamic programming based algorithm. By only including files whose minimum redundancy is smaller than the original length ($\log r_n(b, k) < 8n$), we are left with 122 files. Figure 4.6 shows the histogram of the best (b, k) pairs for these 122 files. It can be seen that for majority of the files studied, viewing edits as k-substring edits for $k > 1$ minimizes the redundancy. In particular, for $k = 10$, a group of pairs of files differ by $b = 66$ k-substring edits. A more careful analysis reveals that the files in the new version 5.1 added a claim of copyrights compared to the corresponding old version 5.0. Comparing Figure 4.5 with Figure 4.6, the best choice of k for bash data is higher for that of DNA. It represents that the editing of files is prone of bursty manners.

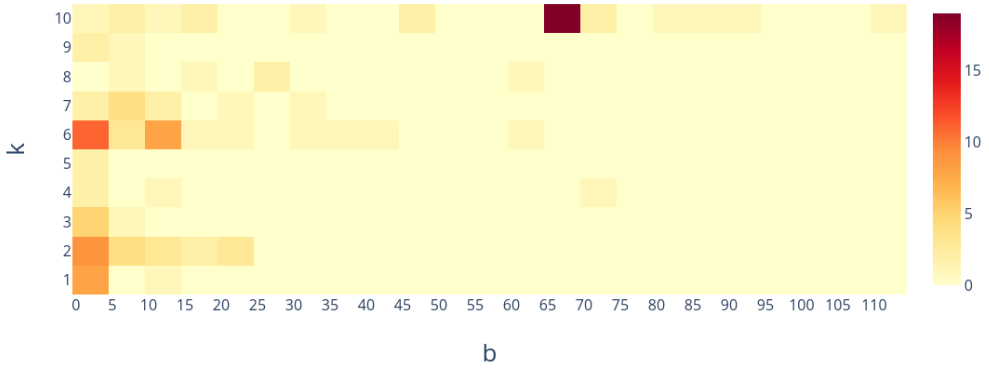


Figure 4.6: Histogram of optimal (b, k) values for the bash dataset.

Based on Figure 4.5 and Figure 4.6, codes correcting burst errors achieve lower redundancy compared to codes correcting independent indels. To further support the results, Table 4.3 presents the average redundancy of indel-correcting codes ($k = 1$) and burst-correcting codes ($k \geqslant 1$) to correct a symbol over two datasets, respectively. In this experiment, suppose a dataset contains N files with file size $\{n_1, \cdots, n_N\}$, we define the average redundancy of correcting the datasets as $\sum_{i=1}^{N} \log_q r_{n_i}(b_i, k_i) / \sum_i^N n_i$, where (b_i, k_i) is the pair for the ith file. Based on the numerical results, for both datasets, the burst-error-correcting codes ($k \geqslant 1$) achieve a lower redundancy compared to the indels-error-correcting codes ($k = 1$).

	Bash dataset		Nanopore sequencing dataset	
k	$k \geqslant 1$	$k = 1$	$k \geqslant 1$	$k = 1$
average $\frac{\log_q r_n(b,k)}{n}$	0.30	0.62	0.84	1.06

Table 4.3: Comparison of redundancy of burst-error correcting codes with $k \geqslant 1$ and independent indel-error-correcting codes with $k = 1$.

4.4 Challenges of correcting a k-substring edit

Given a constant k, this section focuses on constructing codes of length N for correcting a k-substring edit error with redundancy roughly $2\log N$ and polynomial time complexities in both the encoding and decoding processes. Unless otherwise stated, let n represent the length of $(\mathcal{P},\delta)$-dense strings.

Based on Lemma 103, given an input $\boldsymbol{x}\in\mathcal{D}_{\mathcal{P},\delta}(n)$, locating the burst of deletions relies on the number of patterns $n_{\mathcal{P}}(\boldsymbol{x})$ and the relative distances of every two adjacent patterns $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})$. Compared with locating a burst of deletions, locating a k-substring edit is more complicated. Suppose $\boldsymbol{x}\in\mathcal{D}_{\mathcal{P},\delta}(n)$ and $\boldsymbol{y}\in D(\boldsymbol{x})$ is an output generated from $\boldsymbol{x}$ by a k-substring edit. We need to overcome the following challenges. First, when the k-substring edit is a substitution, i.e., $|\boldsymbol{x}|=|\boldsymbol{y}|$, it is possible for both $\boldsymbol{a}_{\mathcal{P}}$ and $n_{\mathcal{P}}$ to remain the same, preventing us from locating the error. Second, even if the substring edit is strict, i.e., $|\boldsymbol{x}|\neq|\boldsymbol{y}|$, there is no guarantee that the changes affecting $\boldsymbol{a}_{\mathcal{P}}$ and $n_{\mathcal{P}}$ will enable us to identify the approximate location of the error. To address these issues, this chapter will extend the previous error-correcting code to correct a strict substring edit and adapt a low-redundancy code to correct a burst of substitutions, respectively.

In order to construct error-correcting codes reaching the GV bound, we have the following corollary, which also summarizes our approach.

Corollary 104. *The code $\mathcal{C}\subseteq\Sigma_2^N$ is capable of correcting a k-substring edit if it can correct either a k-burst substitution or a strict k-substring edit error.*

4.5 Error-correcting code for a strict k-substring edit

Similar to works in [2], [42], given a constant k, this subsection focuses on correcting a strict k-substring edit by first localizing the error and then correcting it in the interval. More specifically, it consists of two steps. First, we extend the error-locating code in Lemma 103 from [2] to locate the strict k-substring edit in an interval of length $L=O((\log n)^2)$ with redundancy roughly $\log n$. Second, a modified *syndrome compression code* [75], [93] is designed to correct the error in the specific interval with redundancy roughly $O(\log\log n)$.

4.5.1 Locating the error in an interval

This subsection focuses on extending the codes in Lemma 103 to locate a strict k-substring edit since it will affect at least one of $n_{\mathcal{P}}(\boldsymbol{x})$ and $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})$.

Lemma 105. *Given $\mathcal{P}=0^k1^k$ and $\delta=k2^{2k+1}\lceil\log n\rceil$, let $\boldsymbol{x}\in\mathcal{D}_{\mathcal{P},\delta}(n)\cap\Sigma_2^n$ be a $(\mathcal{P},\delta)$-dense string and $\boldsymbol{y}\in D(\boldsymbol{x})$. Then a k-substring edit does not create nor destroy more than two* adjacent *patterns $\mathcal{P}$ in $\boldsymbol{x}$, i.e., $n_{\mathcal{P}}(\boldsymbol{y})\in[n_{\mathcal{P}}(\boldsymbol{x})-2,n_{\mathcal{P}}(\boldsymbol{x})+2]$.*

Proof. Let $\boldsymbol{x}=\boldsymbol{uvw}$ and $\boldsymbol{y}=\boldsymbol{uv'w}$. Observe that

$$n_{\mathcal{P}}(\boldsymbol{u})+n_{\mathcal{P}}(\boldsymbol{w})\overset{\text{(a)}}{\leqslant}n_{\mathcal{P}}(\boldsymbol{x})\overset{\text{(b)}}{\leqslant}n_{\mathcal{P}}(\boldsymbol{u})+n_{\mathcal{P}}(\boldsymbol{w})+2,$$

$$n_{\mathcal{P}}(\boldsymbol{u}) + n_{\mathcal{P}}(\boldsymbol{w}) \overset{\text{(c)}}{\leqslant} n_{\mathcal{P}}(\boldsymbol{y}) \overset{\text{(d)}}{\leqslant} n_{\mathcal{P}}(\boldsymbol{u}) + n_{\mathcal{P}}(\boldsymbol{w}) + 2,$$

where the upper bounds holds because $|\boldsymbol{v}|, |\boldsymbol{v}'| \leqslant k$ and the length of the pattern is $2k$. The upper bound on $n_{\mathcal{P}}(\boldsymbol{y})$ follows from (d) and (a), and the lower bound from (c) and (b). □

Then we have the following corollary.

Corollary 106. *Let $\boldsymbol{x} \in \mathcal{D}_{\mathcal{P},\delta}(n)$ and $\boldsymbol{y} \in D(\boldsymbol{x})$. Then $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{y})$ can be generated from $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})$ as a result of a 3-substring-edit.*

According to the changes of $n_{\mathcal{P}}(\boldsymbol{x})$ and $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})$, we extend the code in Bitar et al. work [2] as the following construction that helps to locate a strict k-substring edit occurring in the $(\mathcal{P}, \delta)$-dense string for $\boldsymbol{x} \in \mathcal{D}_{\mathcal{P},\delta}(n)$. Compared to localized deletions only reducing the length, the following modification can deal with the case of increasing the length and locating a strict k-substring edit in a wider interval.

Construction 107. *Given $0 \leqslant c_1 \leqslant 4$ and $0 \leqslant c_2 < 7n$, let*

$$\begin{aligned}\mathcal{C}_{loc}(c_1, c_2) = \{&\boldsymbol{x} \in \Sigma_2^n \cap \mathcal{D}_{\mathcal{P},\delta}(n), n_{\mathcal{P}}(\boldsymbol{x}) = c_1 \bmod 5, \\ &VT(\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})) = c_2 \bmod 7n\}.\end{aligned}$$

Then we have the following lemma.

Lemma 108. *Let k be a constant. Given $\boldsymbol{x} \in \mathcal{C}_{loc}(c_1, c_2)$, a strict k-substring edit occurring in $\boldsymbol{x}$ can be located in a substring of $\boldsymbol{x}$ with length $L = O(\delta^2) = O((\log n)^2)$.*

Proof. The lemma is proved by adapting a similar method from the work [2]. The key idea is to construct a monotonic function with respect to the index of the first message block, partitioned by patterns, that is affected by the strict k-substring edit. Then we can find a range of the indices, leading to an interval in which the strict k-substring edit occurs.

Given $\boldsymbol{x} \in \Sigma_2^n \cap \mathcal{D}_{\mathcal{P},\delta}(n)$ and $\boldsymbol{y} \in D(\boldsymbol{x})$, we get $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})$ and $n_{\mathcal{P}}(\boldsymbol{x})$ from $\boldsymbol{x}$ since $x \in \mathcal{C}_{loc}(c_1, c_2)$. Similarly, $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{y})$ and $n_{\mathcal{P}}(\boldsymbol{y})$ from $\boldsymbol{y}$. Since $\boldsymbol{x} \in \mathcal{D}_{\mathcal{P},\delta}(n)$, we have $2k \leqslant a_{\mathcal{P}}(\boldsymbol{x})_i \leqslant \delta$ for $i \geqslant 2$.

In the following, we analyze the changes from $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})$ to $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{y})$. Based on Lemma 106, the vector $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{y})$ can be considered generating from $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})$ by replacing a substring $\boldsymbol{u}$ in $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})$ to another substring $\boldsymbol{v}$ in $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{y})$, i.e.,

$$\begin{aligned}\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x}) &= (a_{\mathcal{P}}(\boldsymbol{x})_{[1,t)}, \boldsymbol{u}, a_{\mathcal{P}}(\boldsymbol{x})_{[e_1, n_{\mathcal{P}}(\boldsymbol{x})]}) \\ &= (a_{\mathcal{P}}(\boldsymbol{y})_{[1,t)}, \boldsymbol{u}, a_{\mathcal{P}}(\boldsymbol{y})_{[e_2, n_{\mathcal{P}}(\boldsymbol{y})]}), \\ \boldsymbol{a}_{\mathcal{P}}(\boldsymbol{y}) &= (a_{\mathcal{P}}(\boldsymbol{x})_{[1,t)}, \boldsymbol{v}, a_{\mathcal{P}}(\boldsymbol{x})_{[e_1, n_{\mathcal{P}}(\boldsymbol{x})]}) \\ &= (a_{\mathcal{P}}(\boldsymbol{y})_{[1,t)}, \boldsymbol{v}, a_{\mathcal{P}}(\boldsymbol{y})_{[e_2, n_{\mathcal{P}}(\boldsymbol{y})]}).\end{aligned}$$

where $0 \leqslant |\boldsymbol{u}|, |\boldsymbol{v}| \leqslant 3$, $t \leqslant e_1, e_2 \leqslant t+3$. Note that $a_{\mathcal{P}}(\boldsymbol{y})_i$ satisfies $2k \leqslant a_{\mathcal{P}}(\boldsymbol{y})_i \leqslant 3\delta + k < 4\delta$, where the lower bound is obtained since $\mathcal{P}$ has length $2k$ for $i \geqslant 2$. Furthermore, based on Lemma 105,

we have $|n_{\mathcal{P}}(\boldsymbol{y}) - n_{\mathcal{P}}(\boldsymbol{x})| \leqslant 2$ and $||\boldsymbol{u}| - |\boldsymbol{v}|| \leqslant 2$. Therefore, the length of $a_{\mathcal{P}}(\boldsymbol{y})_i$ is upper bounded by $3\delta + k$ after breaking two patterns and inserting a substring of length upper bounded by k.

Let $d = |\boldsymbol{u}| - |\boldsymbol{v}|$. Then $d \in [-2, 2]$ can be uniquely computed from $\boldsymbol{y}$, i.e., $d = n_{\mathcal{P}}(\boldsymbol{y}) - c_1 \bmod 5$. Let $k' = |\boldsymbol{x}| - |\boldsymbol{y}|$, then we have $k' = \sum_{i=t}^{e_1-1} a_{\mathcal{P}}(\boldsymbol{x})_i - \sum_{i=t}^{e_2-1} a_{\mathcal{P}}(\boldsymbol{y})_i$, where $0 < |k'| \leqslant k$. Note that $k' \neq 0$ for a strict k-substring edit.

Then we have

$$\begin{aligned}
Z :=& VT(\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{y})) - VT(\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})) \\
=& d \sum_{i=e_2}^{n_{\mathcal{P}}(\boldsymbol{y})} a_{\mathcal{P}}(\boldsymbol{y})_i - t\left(\sum_{i=t}^{e_1-1} a_{\mathcal{P}}(\boldsymbol{x})_i - \sum_{i=t}^{e_2-1} a_{\mathcal{P}}(\boldsymbol{y})_i\right) \\
& - \sum_{i=t+1}^{e_1-1} (i-t) a_{\mathcal{P}}(\boldsymbol{x})_i + \sum_{i=t+1}^{e_2-1} (i-t) a_{\mathcal{P}}(\boldsymbol{y})_i \\
=& d \sum_{i=t+3}^{n_{\mathcal{P}}(\boldsymbol{y})} a_{\mathcal{P}}(\boldsymbol{y})_i - tk' + E,
\end{aligned}$$

where $0 < |k'| \leqslant k$ and E is

$$E = d \sum_{i=e_2}^{t+2} a_{\mathcal{P}}(\boldsymbol{y})_i - \sum_{i=t+1}^{e_1-1} (i-t) a_{\mathcal{P}}(\boldsymbol{x})_i + \sum_{i=t+1}^{e_2-1} (i-t) a_{\mathcal{P}}(\boldsymbol{y})_i.$$

However, the range of E will be modified since $a_{\mathcal{P}}(\boldsymbol{y})_i$ has a different range. Since $e_2 \leqslant t + 3$, the total components of the first and the last summation are at most three each with coefficients $\max(|d|, e_2 - t - 1) \leqslant 2$. Then we have $|E| \leqslant 3 \cdot 2 \cdot \max(a_{\mathcal{P}}(\boldsymbol{y})_i) < 24\delta$. Then the difference of VT check sum satisfies

$$Z = d \sum_{i=t+3}^{n_{\mathcal{P}}(\boldsymbol{y})} a_{\mathcal{P}}(\boldsymbol{y})_i - tk' + E, |E| < 24\delta.$$

where $0 < |k'| \leqslant k$ results from a strict k-substring edit compared to $k \geqslant k' > 0$ for a burst deletion. Therefore, we have $-3n - 24\delta < Z < 3n + 24\delta$. Since $7n > 6n + 48\delta + 1$ and $VT(a_{\mathcal{P}}(\boldsymbol{x})) = c_2 \bmod 7n$, the integer $Z \in [-3n - 24\delta + 1, 3n + 24\delta - 1]$ can be uniquely obtained based on $Z = VT(a_{\mathcal{P}}(\boldsymbol{y})) - c_2 \bmod 7n$.

In the following, we define a function

$$H(t) := d \sum_{i=t+3}^{n_{\mathcal{P}}(\boldsymbol{y})} a_{\mathcal{P}}(\boldsymbol{y})_i - tk'.$$

Our task is to prove that $H(t)$ is a strictly monotonic function with respect to t for $0 < |k'| \leqslant k$, not only $k \geqslant k' > 0$:

- Firstly, suppose $0 < k' \leqslant k$. If $d \geqslant 0$, $H(t)$ is a strictly decreasing function of t. If $d < 0$, then by the function $a_{\mathcal{P}}(\boldsymbol{y})_i \geqslant 2k > k'$, then $H(t)$ is a strictly increasing function of t.
- Secondly, suppose $-k \leqslant k' < 0$. If $d \leqslant 0$, $H(t)$ is a strictly increasing function of t. If $d > 0$,

since $a_{\mathcal{P}}(\boldsymbol{y})_i \geqslant 2k > -k'$, $H(t)$ is a strictly decreasing function of t.

Given Z, k', and d, the task is to locate t in the set

$$I = \{t : H(t) \in [Z - 24\delta, Z + 24\delta]\}.$$

For a monotonic function $H(t)$, there are at most $48\delta + 1$ choices of t. Since $a_{\mathcal{P}}(\boldsymbol{y})_i < 4\delta$, we can locate the strict k-substring edit in an interval of length at most $192\delta^2 + 4\delta = O(\delta^2) = O((\log n)^2)$. The time complexity to compute $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{y})$, $n_{\mathcal{P}}(\boldsymbol{y})$, and the interval is $O(n)$. □

4.5.2 Correcting the error in an interval

Suppose a strict k-substring edit is located in an interval of lenth $L = O(\delta^2)$. Next, we present error-correcting codes that can correct the strict k-substring edit in the specific interval with length L.

Similar to the work [93], we generate two sets of blocks of length $2L$ by partitioning $\boldsymbol{x} \in \mathcal{D}_{\mathcal{P},\delta}(n)$. More specifically, given $\hat{N} = n/2L$, let $S_e = (\boldsymbol{s}_{e1}, \boldsymbol{s}_{e2}, \cdots, \boldsymbol{s}_{e\hat{N}})$ and $S_o = (\boldsymbol{s}_{o1}, \boldsymbol{s}_{o2}, \cdots, \boldsymbol{s}_{o(\hat{N}-1)})$ denote the set of even and odd blocks respectively, where $\boldsymbol{s}_{ei} = \boldsymbol{x}_{[2(i-1)L+1:2iL]}$ for $i \in [\hat{N}]$ and $\boldsymbol{s}_{oi} = \boldsymbol{x}_{[(2i-1)L+1:(2i+1)L]}$ for $i \in [\hat{N} - 1]$.

Since the k-substring edit is located in a specific interval with the length bounded by L, then it will occur in at least one block of either S_e and S_o. In the following, we apply a modified syndrome compression code [75] to correct a strict k-substring edit in a length-$2L$ string.

Based on Theorem 102, for $\boldsymbol{u} \in \Sigma_2^{2L}$ and a labeling function f over $B(\boldsymbol{u})$, the decoder can recover $\boldsymbol{u}$ by $\boldsymbol{v} \in D(\boldsymbol{u})$, a, and $f(\boldsymbol{u}) \bmod a$. Furthermore, a complex labeling function f does not affect the redundancy of the code since the redundancy $2 \log a$ is affected by the size of $B(\boldsymbol{u})$. Since a strict k-substring edit can be viewed as a k-burst deletion followed by a k-burst insertion, we introduce a labeling function that can correct at most $2k$ insertions, deletions, and substitutions. The following theorem introduces a function from the work [74] to correct at most $t = 2k$ insertions, deletions, and substitutions.

Theorem 109 (cf. [74]). *Given a constant k, $t = 2k$, and $L = O((\log n)^2)$, There exists a labeling function $g : \Sigma_2^{2L} \to \Sigma_{2^{\mathcal{R}(t,2L)}}$ such that for any two distinct strings $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ confusable under at most t insertions, deletions, and substitutions, we have $g(\boldsymbol{s}_1) \neq g(\boldsymbol{s}_2)$, where $\mathcal{R}(t, 2L) = ((t^2 + 1)(2t^2 + 1) + 2t^2(t - 1)) \log 2L + o(\log 2L) = O(\log \log n) + o(\log \log n)$.*

Based on Lemma 101, given $\boldsymbol{u} \in \Sigma_2^{2L}$, the size of the confusable set $B(\boldsymbol{u})$ satisfies $\|B(\boldsymbol{u})\| < (2L + k)^2(k + 1)^4 2^{2k}$. Then for each $\boldsymbol{s}_{ei} \in \Sigma_2^{2L}$ and $\boldsymbol{w}_{ei} \in B(s_{ei})$, there exists an integer a_{ei} such that $g(\boldsymbol{s}_{ei}) \not\equiv g(\boldsymbol{w}_{ei}) \bmod a_{ei}$ for $i \in [\hat{N}]$, where $a_{ei} \leqslant 2^{\log \|B(\boldsymbol{s}_{ei})\| + o(\log L)}$. The same property holds for each $\boldsymbol{s}_{oi} \in \Sigma_2^{2L}$ for $i \in [\hat{N} - 1]$.

Based on the two sets of messages S_e, S_o, we have the following construction for a k-substring edit.

Construction 110. *Let $\boldsymbol{\beta} = (\beta_1, \beta_2, \cdots, \beta_6)$. Given $\boldsymbol{x} \in \mathcal{C}_{loc}(\beta_1, \beta_2)$ with length n, we generate two sets of message blocks S_e and S_o. Let $\beta_3, \cdots, \beta_6 < \alpha$. Then we have*

$$\begin{aligned}
&\mathcal{C}_{strict}(\boldsymbol{\beta}, \alpha) = \{\boldsymbol{x} \in \mathcal{C}_{loc}(\beta_1, \beta_2), \\
&\sum_{i=1}^{\hat{N}} a_{ei} = \beta_3 \bmod \alpha, \sum_{i=1}^{\hat{N}} (g(\boldsymbol{s}_{ei}) \bmod a_{ei}) = \beta_4 \bmod \alpha, \\
&\sum_{i=1}^{\hat{N}-1} a_{oi} = \beta_5 \bmod \alpha, \sum_{i=1}^{\hat{N}-1} (g(\boldsymbol{s}_{oi}) \bmod a_{oi}) = \beta_6 \bmod \alpha.\}
\end{aligned}$$

where α satisfies $\alpha \geqslant (2L+k)^2(k+1)^4 2^{2k} 2^{o(\log L)} > \max(a_{e1}, a_{o1} \cdots, a_{e(\hat{N}-1)}, a_{o(\hat{N}-1)}, a_{eN})$.

Note that a similar construction appeared recently in [93], [94] for burst deletions. However, our construction includes a more powerful error-locating code and modified modulus so that our code can correct a strict k-substring edit.

Lemma 111. *Given a constant k, the error-correcting code $\mathcal{C}_{strict}(\boldsymbol{\beta}, \alpha)$ in Construction 110 can correct a strict k-substring edit error with the redundancy of roughly $\log n + 16 \log\log n + o(\log\log n)$ bits.*

Proof. Let $\boldsymbol{x} \in \mathcal{C}_{strict}(\boldsymbol{\beta}, \alpha)$ and $\boldsymbol{z} \in D(\boldsymbol{x})$ with $|\boldsymbol{x}| \neq |\boldsymbol{z}|$. Based on Lemma 108, a strict k-substring edit will be located in an interval of length L.

According to the definition of S_e and S_o, each pair of two blocks in $(\boldsymbol{s}_{oi}, \boldsymbol{s}_{ei})$ and $(\boldsymbol{s}_{oi}, \boldsymbol{s}_{e(i+1)})$ shares a common substring of length L for $i \in [\hat{N}-1]$. Based on the output $\boldsymbol{z}$ and the localized interval, the decoder can generate two sets of blocks, i.e., $S'_e = (\boldsymbol{s}'_{e1}, \boldsymbol{s}'_{e2}, \cdots, \boldsymbol{s}'_{eN})$ and $S'_o = (\boldsymbol{s}'_{o1}, \boldsymbol{s}'_{o2}, \cdots, \boldsymbol{s}'_{o(N-1)})$, similar to S_e and S_o. Then for at least one of two recovered sets S'_e, S'_o at the decoder, saying S'_e, only one block saying $\boldsymbol{s}'_{ei}$ for $i \in [N]$ has a different block length and contains the whole interval of length L where the strict k-substring edit occurs. Then $\boldsymbol{s}'_{ei} \in D(\boldsymbol{s}_{ei})$ and the other blocks in S'_e are error-free. For all error-free blocks $\boldsymbol{s}'_{ej}$ with $j \neq i$, we can recover a_{ej} and $g(\boldsymbol{s}_{ej})$. Then we can uniquely recover a_{ei} and $g(\boldsymbol{s}_{ei}) \bmod a_{ei}$. Then we can recover $\boldsymbol{s}_{ei}$ based on $\boldsymbol{s}'_{ei}$, a_{ei}, and $g(\boldsymbol{s}_{ei}) \bmod a_{ei}$. As a result, we can recover $\boldsymbol{x}$ based on S'_e.

Based on Construct 110, there exists some code $\mathcal{C}_{strict}(\boldsymbol{\beta}, \alpha)$ such that $\|\mathcal{C}_{strict}(\boldsymbol{\beta}, \alpha)\| \geqslant \frac{2^{n-1}}{\beta_1 \beta_2 \alpha^4}$. Let $\alpha = (2L+k)^2(k+1)^4 2^{2k} 2^{o(\log L)}$ with $L = O((\log n)^2)$, the redundancy of the code is

$$\begin{aligned}
n - \log \|\mathcal{C}_{strict}(\boldsymbol{\beta}, \alpha)\| &\leqslant 1 + 4\log\alpha + \log 7n + \log 5 \\
&= \log n + 16 \log\log n + o(\log\log n).
\end{aligned}$$

□

4.6 Error-correcting code for a k-burst substitution

In this subsection, we present a code that can correct a k-burst substitution error.

Construction 112 (cf. [3], Fire code)**.** *Let $g_0(x)$ be an irreducible polynomial of degree $m \geqslant k$ that does not divide $x^{2k-1}-1$. Then, there exists a linear cyclic code (called Fire code) of length $n_1 = \mathrm{LCM}(2k-1, 2^m-1)$ with the* generator polynomial $g(x) = (x^{2k-1}-1)g_0(x)$ *and* $\deg(g(x)) = m+2k-1$. *Then the Fire code $\mathcal{C}_F$ can be represented as $[n_1, n]$ code with the codeword length n_1 and the dimension $n = n_1 - (m+2k-1)$.*

Theorem 113 (cf. [3])**.** *The Fire code $\mathcal{C}_F$ can correct a single k-burst substitution.*

Then the redundancy of the Fire code can be presented by the following lemma.

Lemma 114. *Given a constant k, the Fire code $\mathcal{C}_F$ corrects a k-burst substitution with the redundancy roughly* $\log n + o(\log n)$.

Proof. Based on Construction 112 and the definition of the redundancy of an error-correcting code, the redundancy of the Fire code is

$$r(\mathcal{C}_F) = n_1 - \log_2 \|\mathcal{C}_F\| = m + 2k - 1.$$

Given $\boldsymbol{x} \in \Sigma_2^n$, since $m \geqslant k$, the length of Fire code satisfies

$$2^m - 1 \leqslant n_1 = n + m + 2k - 1 \leqslant (2k-1)(2^m-1).$$

Therefore, we have $m = \log n_1 + o(\log n_1) = \log n + o(\log n) > k$ as $n \to \infty$. Therefore, the redundancy of the Fire code is roughly $\log n + o(\log n)$ when k is a constant. □

Hence, given $\boldsymbol{x} \in \Sigma_2^n$, there exists a function $h_F(\boldsymbol{x})$ of length roughly $\log n + o(\log n)$ such that $(\boldsymbol{x}, h_F(\boldsymbol{x})) \in \mathcal{C}_F$ is capable of correcting a k-burst substitution.

4.7 Combined error-correcting codes

Based on Lemma 104, given $\boldsymbol{x} \in \mathcal{D}_{\mathcal{P},\delta}(n) \cap \Sigma_2^n$, the receiver can correct a k-substring edit from $\boldsymbol{y} \in D(\boldsymbol{x})$ if $h_F(\boldsymbol{x})$ and $(\boldsymbol{\beta}, \alpha)$ are sent to the receiver by an error-free channel. For simplicity, let $\boldsymbol{r}_{\boldsymbol{x}} := (h_F(\boldsymbol{x}), \boldsymbol{\beta}, \alpha)$ be a binary representation of the data. Then each codeword of the final error-correcting codes can be generated by concatenating two parts, i.e., $(\boldsymbol{x}, \boldsymbol{r}_{\boldsymbol{x}})$. Furthermore, we may add a buffer between $\boldsymbol{x}$ and $\boldsymbol{r}_{\boldsymbol{x}}$ such that a k-substring edit affects either $\boldsymbol{x}$ or $\boldsymbol{r}_{\boldsymbol{x}}$. Finally, We also need another function that can detect or correct a k-substring edit occurring in $\boldsymbol{r}_{\boldsymbol{x}}$. We start by finding a suitable buffer $\boldsymbol{b}_{\boldsymbol{x}}$.

Since a k-substring edit should not affect both $\boldsymbol{x}$ and $\boldsymbol{r}_{\boldsymbol{x}}$, the length of the buffer satisfies $|\boldsymbol{b}_{\boldsymbol{x}}| > k$. The buffer in the following lemma helps distinguish whether the strict k-substring edit affects $\boldsymbol{x}$ or $\boldsymbol{r}_{\boldsymbol{x}}$.

Lemma 115. *Given a string $\boldsymbol{w} = \boldsymbol{x}\boldsymbol{b}_{\boldsymbol{x}}\boldsymbol{u}$ with $\boldsymbol{x} \in \Sigma_2^n$ and $\boldsymbol{u} \in \Sigma_2^*$. A buffer $\boldsymbol{b}_{\boldsymbol{x}} = 1^{k+1}0^{k+1}1^{k+1}$ can distinguish whether a strict k-substring edit affect either $\boldsymbol{x}$ or $\boldsymbol{u}$.*

Proof. Let $\boldsymbol{w} = \boldsymbol{x}\boldsymbol{b}_{\boldsymbol{x}}\boldsymbol{u}$ with $\boldsymbol{x} \in \Sigma_2^n$ and $\boldsymbol{u} \in \Sigma_2^*$. Suppose $\boldsymbol{b}_{\boldsymbol{x}} = 1^{k+1}0^{k+1}1^{k+1} = \boldsymbol{w}_{[n+1:n+3k+3]}$. Let $\boldsymbol{z} \in D(\boldsymbol{w})$. Since a strict k-substring edit is considered, we have $|\boldsymbol{z}| \neq |\boldsymbol{w}|$. Furthermore, let $k' = |\boldsymbol{w}| - |\boldsymbol{z}|$. Then we analyze the effect of a strict k-substring edit in the following cases.

- If $0 < k' \leqslant k$, we focus on the substring $\boldsymbol{z}_{[n+1-k',n+3k+3]}$.
  - If $\boldsymbol{z}_{[n+1-k',n+3k+3-k']} = 1^{k+1}0^{k+1}1^{k+1}$ or if $\boldsymbol{z}_{[n+k+2-k',n+3k+3-k']} = 0^{k+1}1^{k+1}$, the strict k-substring edit does not affect $\boldsymbol{u}$. Furthermore, $\boldsymbol{z}_{[1:n-k']} \in D(\boldsymbol{x})$.
  - If $\boldsymbol{z}_{[n+1,n+3k+3]} = 1^{k+1}0^{k+1}1^{k+1}$ or if $\boldsymbol{z}_{[n+1,n+2k+2]} = 1^{k+1}0^{k+1}$, the strict k-substring edit does not affect $\boldsymbol{x}$.
  - Otherwise, the strict k-substring edit only affects the buffer, does not affect both $\boldsymbol{x}$ and $\boldsymbol{u}$.
- If $-k \leqslant k' < 0$, we focus on the substring $\boldsymbol{z}_{[n+1,n+3k+3-k']}$.
  - If $\boldsymbol{z}_{[n+1,n+3k+3]} = 1^{k+1}0^{k+1}1^{k+1}$ or if $\boldsymbol{z}_{[n+1,n+2k+2]} = 1^{k+1}0^{k+1}$, the strict k-substring edit does not affect $\boldsymbol{x}$.
  - If $\boldsymbol{z}_{[n+1-k',n+3k+3-k']} = 1^{k+1}0^{k+1}1^{k+1}$ or if $\boldsymbol{z}_{[n+k+2-k',n+3k+3-k']} = 0^{k+1}1^{k+1}$, the strict k-substring edit does not affect $\boldsymbol{u}$. Furthermore, $\boldsymbol{z}_{[1:n-k']} \in D(\boldsymbol{x})$.
  - Otherwise, the strict k-substring edit only affect the buffer, does not affect both $\boldsymbol{x}$ and $\boldsymbol{u}$.

Hence, the buffer $\boldsymbol{b}_{\boldsymbol{x}} = 1^{k+1}0^{k+1}1^{k+1}$ can help distinguish whether a strict k-substring edit affects either $\boldsymbol{x}$ or $\boldsymbol{u}$. □

Then a burst-substitution-detecting function $\mathcal{E}_1$ suffices to decode $\boldsymbol{x}$ if it can detect an k-burst substitution in $\boldsymbol{r}_{\boldsymbol{x}}$.

Lemma 116. *Given a constant k and $\boldsymbol{b}_{\boldsymbol{x}} = 1^{k+1}0^{k+1}1^{k+1}$, a burst-substitution-detecting function $\mathcal{E}_1$ in $\boldsymbol{x}\boldsymbol{b}_{\boldsymbol{x}}\boldsymbol{r}_{\boldsymbol{x}}\mathcal{E}_1(\boldsymbol{r}_{\boldsymbol{x}})$ is sufficient to decode $\boldsymbol{x}$ for a k-substring edit.*

Proof. Recall that since the length of the buffer is larger than k, a k-substring edit will not affect $\boldsymbol{x}$ and $\boldsymbol{r}_{\boldsymbol{x}}\mathcal{E}_1(\boldsymbol{r}_{\boldsymbol{x}})$ simultaneously. Then we present how a burst-substitution-detecting function is sufficient to decode $\boldsymbol{x}$. Given $\boldsymbol{w} = \boldsymbol{x}\boldsymbol{b}_{\boldsymbol{x}}\boldsymbol{r}_{\boldsymbol{x}}\mathcal{E}_1(\boldsymbol{r}_{\boldsymbol{x}})$, let $\boldsymbol{z} = D(\boldsymbol{w})$. Furthermore, let $k' = |\boldsymbol{w}| - |\boldsymbol{z}|$. Then we decode $\boldsymbol{x}$ in the following process.

- Suppose $\boldsymbol{x}\boldsymbol{b}_{\boldsymbol{x}}\boldsymbol{r}_{\boldsymbol{x}}\mathcal{E}_1(\boldsymbol{r}_{\boldsymbol{x}})$ suffers a strict k-substring edit and $k' \neq 0$. Based on Lemma 115, we can distinguish whether $\boldsymbol{x}$ is affected. If $\boldsymbol{x}$ is error-free, we are done. If $\boldsymbol{x}$ is affected by a strict k-substring edit, then $\boldsymbol{r}_{\boldsymbol{x}}$ is error-free. Since $\boldsymbol{z}_{[1:n-k']} \in D(\boldsymbol{x})$, then $\boldsymbol{x}$ can be recovered from $\boldsymbol{z}_{[1:n-k']}$ and $(\boldsymbol{\beta}, \alpha)$ in $\boldsymbol{r}_{\boldsymbol{x}}$.
- Suppose $\boldsymbol{x}\boldsymbol{b}_{\boldsymbol{x}}\boldsymbol{r}_{\boldsymbol{x}}\mathcal{E}_1(\boldsymbol{r}_{\boldsymbol{x}})$ suffers a k-burst substitution and $k' = 0$. Then we decode $\boldsymbol{x}$ in following cases:

– If $\boldsymbol{z}_{[n+1,n+3k+3]} = 1^{k+1}0^{k+1}1^{k+1}$, the burst substitution affects either $\boldsymbol{x}$ or $\boldsymbol{r}_{\boldsymbol{x}}\mathcal{E}_1(\boldsymbol{r}_{\boldsymbol{x}})$. If the burst-substitution-detecting code $\mathcal{E}_1$ does not detect an error in $\boldsymbol{r}_{\boldsymbol{x}}\mathcal{E}_1(\boldsymbol{r}_{\boldsymbol{x}})$, then $\boldsymbol{x}$ is affected. Then we can decode $\boldsymbol{x}$ from $\boldsymbol{z}_{[1:n]} \in D(\boldsymbol{x})$ and $h_F(\boldsymbol{x})$ in $\boldsymbol{r}_{\boldsymbol{x}}$. If an error is detected in $\boldsymbol{r}_{\boldsymbol{x}}\mathcal{E}_1(\boldsymbol{r}_{\boldsymbol{x}})$, $\boldsymbol{x} = \boldsymbol{z}_{[1:n]}$ is error-free.

– $\boldsymbol{z}_{[n+1,n+3k+3]} \neq 1^{k+1}0^{k+1}1^{k+1}$ but $\boldsymbol{z}_{[n+1,n+2k+2]} = 1^{k+1}0^{k+1}$, $\boldsymbol{x}$ is error-free.

– $\boldsymbol{z}_{[n+1,n+3k+3]} \neq 1^{k+1}0^{k+1}1^{k+1}$ but $\boldsymbol{z}_{[n+k+2,n+3k+3]} = 0^{k+1}1^{k+1}$, then $\boldsymbol{z}_{[1:n]} \in D(\boldsymbol{x})$ can be generated from $\boldsymbol{x}$ by a k-burst substitution. Thus, we can decode $\boldsymbol{x}$ from $\boldsymbol{z}_{[1:n]} \in D(\boldsymbol{x})$ and $h_F(\boldsymbol{x})$.

– Otherwise, the k-substring edit only affect the buffer, $\boldsymbol{x}$ is error-free.

□

The simplest burst-substitution-detecting function is a parity-checking function. Given $\boldsymbol{r}_{\boldsymbol{x}}$, let $L_1 = |\boldsymbol{r}_{\boldsymbol{x}}| > k$. Then we partition $\boldsymbol{r}_{\boldsymbol{x}}$ into $T = \lceil L_1/(k+1) \rceil$ blocks of length $(k+1)$, i.e., $\boldsymbol{w}_j$ for $j \in [T]$, where additional 0s are appended if the last block has less than $(k+1)$ binary symbols. Then the error-detecting function $\mathcal{E}_1 : \Sigma_2^{L_1} \to \Sigma_2^{2k+2}$ appends $\boldsymbol{\gamma}_1$ and $\boldsymbol{\gamma}_2$ in the binary form (each with length $(k+1)$) following $\boldsymbol{r}_{\boldsymbol{x}}$, where

$$\boldsymbol{\gamma}_1 = \left(\sum_{j=1}^{\lceil T/2 \rceil} \boldsymbol{w}_{2j-1} \right) \bmod 2^{k+1},$$
$$\boldsymbol{\gamma}_2 = \left(\sum_{j=1}^{\lfloor T/2 \rfloor} \boldsymbol{w}_{2j} \right) \bmod 2^{k+1}.$$

The final construction is shown below.

Construction 117. *Given a constant k, $\boldsymbol{b}_x = 1^{k+1}0^{k+1}1^{k+1}$, we have a construction $\mathcal{C}_N$ as*

$$\mathcal{C}_N = \{\boldsymbol{x}\boldsymbol{b}_{\boldsymbol{x}}\boldsymbol{r}_{\boldsymbol{x}}\boldsymbol{\gamma}_1\boldsymbol{\gamma}_2 \in \Sigma_2^N, \boldsymbol{x} \in \Sigma_2^n \cap \mathcal{D}_{\mathcal{P},\delta}(n)\},$$

where $\boldsymbol{r}_{\boldsymbol{x}} = (h_F(\boldsymbol{x}), \boldsymbol{\beta}, \alpha)$ and $\boldsymbol{\gamma}_1\boldsymbol{\gamma}_2$ are in binary form generated by $\mathcal{E}_1(\boldsymbol{r}_{\boldsymbol{x}})$ in each codeword $\boldsymbol{x}\boldsymbol{b}_{\boldsymbol{x}}\boldsymbol{r}_{\boldsymbol{x}}\boldsymbol{\gamma}_1\boldsymbol{\gamma}_2 \in \Sigma_2^N$.

Theorem 118. *The error-correcting code $\mathcal{C}_N$ in Construction 117 can correct a k-substring edit with the redundancy of roughly $2\log N + o(\log N)$ bits, where $N = n + 2\log n + o(\log n)$.*

Proof. We first prove that the error-correcting code C_N can correct a k-substring edit. Given $\boldsymbol{w} = \boldsymbol{x}\boldsymbol{b}_{\boldsymbol{x}}\boldsymbol{r}_{\boldsymbol{x}}\boldsymbol{\gamma}_1\boldsymbol{\gamma}_2 \in C_N$ and $\boldsymbol{z} \in D(\boldsymbol{w})$. Let $k' = |\boldsymbol{w}| - |\boldsymbol{z}|$.

- If $k' \neq 0$, the buffer $\boldsymbol{b}_{\boldsymbol{x}}$ helps distinguish whether the strict k-substring edit affect $\boldsymbol{x}$ by Lemma 115. If $\boldsymbol{x}$ is error-free, we are done. If $\boldsymbol{x}$ suffers a substring edit, then $\boldsymbol{r}_{\boldsymbol{x}}$ is error-free and $\boldsymbol{x}$ can be recovered from $\boldsymbol{z}_{[1:n-k']} \in D(\boldsymbol{x})$ and $(\boldsymbol{\beta}, \alpha)$.
- If $k' = 0$. Then $\boldsymbol{\gamma}_1$ and $\boldsymbol{\gamma}_2$ help recover $\boldsymbol{x}$ based on Lemma 116.

Next, we discuss the redundancy of the error-correcting code $\mathcal{C}_N$.

Based on (4.1), the size of the error-correcting code in Construction 117 is $||\mathcal{C}_N|| \geqslant 2^{n-1}$. According to the definition of redundancy, the main task is to explore the length N of the codeword $\boldsymbol{c} = \boldsymbol{x}\boldsymbol{b}_{\boldsymbol{x}}\boldsymbol{r}_{\boldsymbol{x}}\boldsymbol{\gamma}_1\boldsymbol{\gamma}_2$.

Based on the analysis above, the lengths of $\boldsymbol{x}$, $\boldsymbol{b}_{\boldsymbol{x}}$, and $\boldsymbol{r}_{\boldsymbol{x}}$, $\boldsymbol{\gamma}_1\boldsymbol{\gamma}_2$ are n, $3k+3$, $2\log n + o(\log n)$, and $2k+2$, respectively. Therefore, we have $N = n + 2\log n + o(\log n)$. Then the redundancy of the error-correcting code in Construction 117 is roughly

$$N - \log ||\mathcal{C}|| = 2\log n + o(\log n) = 2\log N + o(\log N),$$

where $\log n = \log N + o(\log N)$. □

4.8 Time complexity

This subsection presents that the error-correcting codes $\mathcal{C}_N$ have polynomial time complexities in both encoding and decoding algorithms.

We start with the time complexity of the encoder. Given a constant k, each codeword is generated by four steps:

- First, given a binary message string $\boldsymbol{u}$, an $(\mathcal{P}, \delta)$-dense string $\boldsymbol{x} \in \mathcal{D}_{\mathcal{P},\delta}(n)$ can be generated by Algorithm 2 from Wang et.al [94]. The time complexity is polynomial with respect to n.
- Second, append a buffer $\boldsymbol{b}_{\boldsymbol{x}} = 1^{k+1}0^{k+1}1^{k+1}$ with time complexity $O(1)$.
- Third, produce $\boldsymbol{r}_{\boldsymbol{x}}$ to fight against a k-burst edit by applying the encoders of the linear Fire code [3], the error-locating code in Lemma 108, and the modified syndrome compression codes. For a constant k, the encoders of the Fire code [3] and the error-locating code have polynomial time complexity with respect to n. Furthermore, based on [74], [75], for a constant k, the time complexity of the modified syndrome compression is also polynomial with respect to n.
- Forth, append two parity blocks γ_1 and γ_2 with time complexity $O(n)$.

Therefore, the time complexity of the encoder is polynomial with respect to n.

The decoder consists of detecting the parity check bits, decoding the Fire code [3], locating the error in an interval, and decoding the modified syndrome compression codes. Similarly, for a constant k, the time complexity of the decoder is also polynomial with respect to n. Since $N = n + 2\log n + o(\log n)$, the time complexities of both the encoder and the decoder are polynomial with respect to N.

4.9 Summary

A k-substring edit will have many localized errors such as burst insertions/deletions/substitutions as a special case. Based on two datasets, the statistical hypothesis tests and the codes reaching the GV

bounds show that the substring edits are common errors and substring-edit-correcting codes can achieve a lower redundancy compared to insertion/deletion-error-correcting codes. Furthermore, this chapter constructs error-correcting codes to correct a k-substring edit with the redundancy of roughly $2 \log n$ by using two codes, one error locating based code to correct a strict k-substring edit and one Fire code for a burst of at most k substitutions. Both the encoding and the decoding algorithms have polynomial time complexities.

Chapter 5

Correct deletions over DNA data storage with enzymatic synthesis

5.1 Introduction

As mentioned in Section 1.2.3, the inexpensive enzymatic synthesis will generate multiple DNA strands in parallel. For each round, a noisy (random) number of nucleotides of the same type are appended to a sequence. The deletions (of runs), the dominant errors, occur if no nucleotide is appended in a round. This chapter focuses on constructing error-correcting codes to correct noisy run lengths and deletion errors in the channel model and achieve a writing rate higher than $\log_2 3$ bits per unit time.

The channel model resulting from enzymatic synthesis with noisy run lengths can also be viewed as an insertion-deletion channel with multiple traces. Such a view would benefit from the extensive literature on the topic, including [5], [10], [26], [41], [54], [78]. However, this approach would not be able to take advantage of the specific structure of the deletions and insertions. Namely, each deletion or insertion event modifies a single run and follows specific distributions given the synthesis time for the corresponding round. As each run may be altered by inserting or deleting a large number of symbols, the resulting insertion-deletion probability would be very large. Burst insertion-deletion channels would be more appropriate, but these also ignore the fact that each event is limited to a single run.

To *correct errors arising from noisy run lengths*, similar to [28], the current chapter also uses the PR framework but also proposes error-correcting codes and a decoding algorithm to correct deletions (of runs) in the DNA sequences as well as achieve a writing rate higher than $\log_2 3$ bits per unit time. The code construction contains two codes. Based on [1], we introduce the *block-based Tenegolts q-ary* (BTq) code by concatenating Tenegolts q-ary codes. This code is used to protect against deletions of run by combining with the sequence reconstruction algorithms. Then, by applying a decoding algorithm with sequence reconstruction of alternating sequences, synchronization of run lengths in parallel sequences, and time estimation with the application of the statistical inference, the second *substitution-correcting* (SC) code decodes the information stored in the run lengths[28]. Based on

numerical results, in some settings, the constructed code (including an explicit construction) can decode codewords with the error probability approximating the analytical values while achieving a writing rate higher than $\log_2 3$ bits per unit time. Finally, we discussed a method to set suitable parameters that can achieve the target error probability of decoding both BTq and SC codewords.

The rest of this chapter is organized as follows. Section 5.2 provides the notation and preliminaries. Section 5.3 introduces the channel model. The code construction as well as its rate are presented in Section 5.4, followed by decoding procedures in Section 5.5. Finally, Section 5.6 presents the numerical results while Section 5.7 concludes the section.

5.2 Notation and preliminaries

We start by introducing more notation below apart from that in Section 1.3.3.

Let $\boldsymbol{x} = x_1 x_2 \cdots x_n \in \Sigma_q^n$ be an *alternating string* of length n over Σ_q if $x_i \neq x_{i+1}$ for $i \in [n-1]$. Let S_q^n be the set of alternating strings of length n. Therefore, $S_q^n \subseteq \Sigma_q^n$. Let $[n]$ denote the set $\{1, \ldots, n\}$ and $\mathbb{N}$ and $\mathbb{N}_+$ denote the sets of nonnegative and positive integers, respectively.

A *run* is a maximal substring consisting of a single symbol. For $a \in \Sigma_q, r \in \mathbb{N}$, let $a^r = a \cdots a$ represent a repeated r times. For $\boldsymbol{a} \in \Sigma^n$ and $\boldsymbol{r} \in \mathbb{N}^n$, let $\boldsymbol{x} = \boldsymbol{a}^{\boldsymbol{r}} = a_1^{r_1} a_2^{r_2} \cdots a_n^{r_n}$. If $r_i > 0$ and a_i is different from a_{i-1} and a_{i+1}, then ${a_i}^{r_i}$ is a run in $\boldsymbol{x}$ and r_i is the *run length* of a_i. Note that if $\boldsymbol{a} \in S_q^n$ is alternating and $\boldsymbol{r} \in \mathbb{N}_+^n$ is positive, then $\boldsymbol{x} = \boldsymbol{a}^{\boldsymbol{r}}$ has n runs. We define Ω_q^n as the set consisting of all strings with n runs.

Conversely, given a string $\boldsymbol{x} \in \Omega_q^n$, we can decompose it into its alternating string $\boldsymbol{a} \in S_q^n$ and run length sequence $\boldsymbol{r} \in \mathbb{N}_+^n$ such that $\boldsymbol{a}^{\boldsymbol{r}} = \boldsymbol{x}$. Formally, we define a mapping $\boldsymbol{\psi} : \Omega_q^n \to S_q^n \times \mathbb{N}_+^n$ as

$$\boldsymbol{\psi}(\boldsymbol{x}) = (\boldsymbol{a}, \boldsymbol{r}) = (\boldsymbol{\psi}_1(\boldsymbol{x}), \boldsymbol{\psi}_2(\boldsymbol{x})).$$

For example, given a string $\boldsymbol{x} = 11144331 = 1^3 4^2 3^2 1$, we have $\boldsymbol{a} = \boldsymbol{\psi}_1(\boldsymbol{x}) = 1431$ and $\boldsymbol{r} = \boldsymbol{\psi}_2(\boldsymbol{x}) = 3221$.

Based on [28], given a finite directed and labeled graph $\mathcal{G} = (V, E, \mathcal{L})$ with the set of vertices V, a finite multiset of edges (allowing parallel edges) $E \subseteq V \times V$, and the labels on the edges $\mathcal{L} : E \to \Sigma^+$, the length of an edge $e \in E$ is $l(e) = |\mathcal{L}(e)|$. If all edges have length 1, the graph $\mathcal{G}$ is *ordinary*. Furthermore, a single path $\boldsymbol{s} = e_1 e_2 \cdots e_m$ in $\mathcal{G}$ generates the word $\mathcal{L}(\boldsymbol{s}) = \mathcal{L}(e_1)\mathcal{L}(e_2)\cdots\mathcal{L}(e_m)$, where $e_i \in E$. Given two vertices $v_1, v_2 \in V$ and a string $\boldsymbol{x} \in \Sigma^*$, the graph $\mathcal{G}$ is called *lossless* if we can find at most one path $\boldsymbol{s}$ such that $\mathcal{L}(\boldsymbol{s}) = \boldsymbol{x}$. Based on the graph $\mathcal{G}$, we define a constrained system consisting of all words generated by finite paths in $\mathcal{G}$, $S(\mathcal{G}) = \{\mathcal{L}(\boldsymbol{r}) | \boldsymbol{r}$ is a finite path in $\mathcal{G}\}$. If $\mathcal{G}$ is not ordinary, an equivalent ordinary graph $\mathcal{G}'$ can be constructed by converting each edge $v \xrightarrow{w_1 \ldots w_l} u$ into a path $v \xrightarrow{w_1} v_1 \xrightarrow{w_2} \cdots \xrightarrow{w_{l-1}} v_{l-1} \xrightarrow{w_l} u$ by inserting auxiliary vertices $v_1, \ldots, v_{l-1}$, where $w_i \in \Sigma$.

We next recall the construction of Tenegolts q-ary codes [91] that are capable of correcting a single insertion or deletion (indel) over Σ_q.

Construction 119. *Based on Tenegolts q-ary (Tq) code [91], given integers* $m \geqslant 1$, $0 \leqslant \alpha \leqslant (q-1)$

and $0 \leqslant \beta \leqslant (m-1)$*, we construct the code* $C_{Tq}(\alpha, \beta, m)$ *over* Σ_q *as*

$$C_{Tq}(\alpha,\beta,m) = \Big\{ \boldsymbol{z} \in \Sigma_q^m \Big| \sum_{j=1}^{m} z_j = \alpha \bmod q, \\ \sum_{i=1}^{m} (i-1)\zeta_i(\boldsymbol{z}) = \beta \bmod m \Big\},$$

where the i*-th symbol of* $\boldsymbol{\zeta} : \Sigma_q^m \to \Sigma_2^m$ *is*

$$\zeta_i(\boldsymbol{z}) = \begin{cases} 1, & \text{if } z_i \geqslant z_{i-1}, \\ 0, & \text{if } z_i < z_{i-1}, \end{cases} \qquad i = 2, 3, \ldots, m,$$

with $\zeta_1(\boldsymbol{z}) = 1$.

Given an error-correcting code $\mathcal{C}$, suppose each codeword is encoded from k bits of data and is stored in DNA by T time units of synthesis, we define the *writing rate* of the code as k/T bits per unit time [28].

The key notations used in the chapter are summarized in Table 5.1.

Table 5.1: Key notations in Chapter 5

Notation	Definition
$\Sigma_q = \{0, 1, \cdots, q-1\}$	The alphabet set with q elements
$\|S\|$	The number of elements in the set S
An alternating string	A string $\boldsymbol{x}$ with $\boldsymbol{x}_i \neq \boldsymbol{x}_{i+1}$ for every two consecutive symbols
S_q^n	The set of all length-n alternating sequences
$a^r = a \cdots a$	A run of r consecutive symbols of a with base a and the run length r.
$\boldsymbol{x} = \boldsymbol{a}^{\boldsymbol{r}}$	The sequence $\boldsymbol{x} = \boldsymbol{a}^{\boldsymbol{r}} = \boldsymbol{a}_1^{r_1} \boldsymbol{a}_2^{r_2} \cdots \boldsymbol{a}_n^{r_n}$ with the alternating sequence $\boldsymbol{a}$ and the run length vector $\boldsymbol{r}$
$T_{syn} = \{t_0, t_1, \ldots, t_{L-1}\}$	The set of L different discrete time units
$t_{b_i} \in T_{syn}$	The ith round's synthesis time in T_{syn} with $b_i \in \Sigma_L = \{0, 1, \cdots, L-1\}$

5.3 Channel model for enzymatic DNA synthesis

In this section, we describe the channel model for the enzymatic DNA synthesis in the precision-resolution (PR) framework with deletions of runs.

In enzymatic synthesis N DNA sequences are synthesized in parallel. The synthesis process consists of n rounds, and in each round, a number of nucleotides of the same type, e.g., A, are added to the sequences. Nucleotides added in rounds i and $i+1$ are of different types, so each round adds a run to each sequence. The lengths of the runs are controlled by the duration of the

round. We assume that the number of bases synthesized is a random variable r with distribution[1] $D(t)$, independent of all other events, where t is the associated time spent on the synthesis of the run. Given the noisiness of the run lengths, following the precision-resolution framework of [28], we choose t from a set of discrete times $T_{syn} = \{t_0, t_1, \ldots, t_{L-1}\}$. In this method, N sequences are synthesized simultaneously, all of whom have the same alternating sequence (unless a run is deleted due to no bases being added to the sequence) but the run lengths may differ due to synthesis noise.

We thus model the enzymatic channel in the PR framework as follows: Given two sequences $\boldsymbol{a} \in S_q^n$, $\boldsymbol{b} \in \Sigma_L^n$, and the set T_{syn}, enzymatic DNA synthesis creates N traces $\boldsymbol{x}_j = \boldsymbol{a}^{\boldsymbol{r}_j}$, where the run length r_{ji} for the ith run in the jth trace is determined as $r_{ji} \sim D(t_{b_i})$ with $t_{b_i} \in T_{syn}$.

At the decoder, we determine $\boldsymbol{a}$ and $\boldsymbol{b}$ from $(\boldsymbol{x}_j)_{j=1}^N$, by first finding $\bar{\boldsymbol{a}}^{(j)} = \boldsymbol{\psi}_1(\boldsymbol{x}_j)$ and $\bar{\boldsymbol{r}}_j = \boldsymbol{\psi}_2(\boldsymbol{x}_j)$.

There are two sources of error in this channel. For the moment, assume that we are given $(r_{ji})_{j=1}^N$, from which we must find the value of t_{b_i} and thus b_i. Given that r_{ji} are random quantities, we may decode b_i incorrectly, leading to substitution errors in the codewords $\boldsymbol{b}$. This is the first source of error. The larger the gap between t_k and t_{k+1}, the smaller the probability of this type of error. The second source of error arises from the fact that if $r = 0$, the corresponding run disappears from the trace (this occurs if during synthesis, no base is added to the sequence), leading to deletions of runs. This causes synchronization issues, which may prevent us from determining all of the values $(r_{ji})_{j=1}^N$.

The following example demonstrates the channel model with enzymatic synthesis and possible ways the deletion of runs can affect the alternating and run length sequences, shown in Figure 5.1.

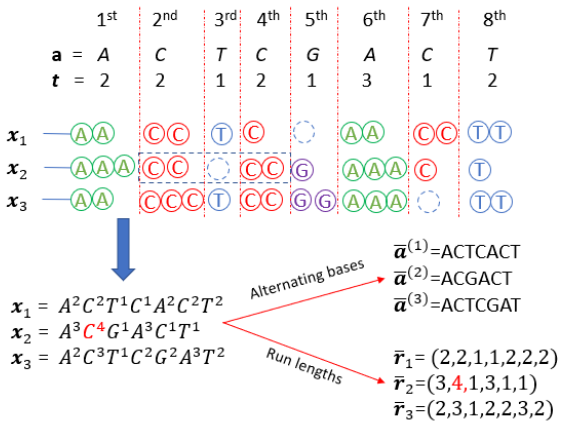


Figure 5.1: An example of the enzymatic synthesis system in the precision-resolution (PR) framework with deletions of runs.

Example 120. *Suppose* $N = 3$ *and* $n = 8$. *Let* $\boldsymbol{a} = ACTCGACT$ *and* $\boldsymbol{t_b} = \boldsymbol{t} = 22121312$. *After* $n = 8$ *rounds of synthesis, three DNA strands are generated in parallel. Suppose the run lengths for three DNA strands are* $\boldsymbol{r}_1 = (2,2,1,1,0,2,2,2)$, $\boldsymbol{r}_2 = (3,2,0,2,1,3,1,1)$, *and* $\boldsymbol{r}_3 =$

[1]In general, this distribution may depend on the previous symbol but for simplicity, we assume the same distribution regardless of the previous base.

$(2,3,1,2,2,3,0,2)$, *where* 0 *represents no base is appended to the corresponding DNA strand. We thus have:*

1. $\boldsymbol{x}_1 = A^2C^2T^1C^2G^0A^2C^2T^2 = A^2C^2T^1C^2A^2C^2T^2$. *Then* $\bar{\boldsymbol{a}}^{(1)} = ACTCACT$ *and* $\bar{\boldsymbol{r}}_1 = (2,2,1,1,2,2,2)$. *Therefore,* $r_{1,5} = 0$ *leads to a deletion in* $\bar{\boldsymbol{a}}^{(1)}$ *compared to* $\boldsymbol{a}$ *and a deletion in the run lengths* $\bar{\boldsymbol{r}}_1$ *compared to* $\boldsymbol{r}_1$.

2. $\boldsymbol{x}_2 = A^3C^2T^0C^2G^1A^3C^1T^1 = A^3C^4G^1A^3C^1T^1$. *Then* $\bar{\boldsymbol{a}}^{(2)} = ACGACT$ *and* $\bar{\boldsymbol{r}}_2 = (3,4,1,3,1,1)$. *Therefore,* $r_{2,3} = 0$ *leads to two deletions in* $\bar{\boldsymbol{a}}^{(2)}$ *compared to* $\boldsymbol{a}$ *and two deletions and a substitution in* $\bar{\boldsymbol{r}}_2$ *compared to* $\boldsymbol{r}_2$. *This occurs because the two runs of* C *are* merged.

3. *Similarly,* $\boldsymbol{x}_3 = A^2C^3T^1C^2G^2A^3C^0T^2 = A^2C^3T^1C^2G^2A^3T^2$. *Then* $\bar{\boldsymbol{a}}^{(3)} = ACTCGAT$ *and* $\bar{\boldsymbol{r}}_3 = (2,3,1,2,2,3,2)$. *Therefore,* $r_{3,7} = 0$ *leads to a deletion in* $\bar{\boldsymbol{a}}^{(3)}$ *compared to* $\boldsymbol{a}$ *and a deletion in the run lengths* $\bar{\boldsymbol{r}}_3$ *compared to* $\boldsymbol{r}_3$.

Based on the example, a single zero run length in the enzymatic synthesis may lead to one or more deletion of (runs) in the output, e.g., the case $r_{1,5} = 0$ *and* $r_{2,3} = 0$. *If the channel with enzymatic synthesis is viewed as an IDS channel [78], the insertion and deletion probabilities may be very high.*

To avoid this challenge, the decoder recovers $\boldsymbol{a}$ *and* $\boldsymbol{t}$ *by using* $\{\bar{\boldsymbol{a}}^{(1)}, \bar{\boldsymbol{a}}^{(2)}, \bar{\boldsymbol{a}}^{(3)}\}$ *and* $\{\bar{\boldsymbol{r}}_1, \bar{\boldsymbol{r}}_2, \bar{\boldsymbol{r}}_3\}$ *respectively. Compared to the* $\boldsymbol{a}$*, the set of output* $\{\bar{\boldsymbol{a}}^{(1)}, \bar{\boldsymbol{a}}^{(2)}, \bar{\boldsymbol{a}}^{(3)}\}$ *suffer multiple deletions. A multiple sequence reconstruction method or error-correcting code can be applied to recover* $\boldsymbol{a}$. *Compared to the run lengths* $\{\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3\}$*, the run lengths* $\{\bar{\boldsymbol{r}}_1, \bar{\boldsymbol{r}}_2, \bar{\boldsymbol{r}}_3\}$ *suffer deletions and substitutions simultaneously. Furthermore, the set of run lengths* $\{\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3\}$ *is not available in the synthesis and retrieval process. Therefore, it is important to propose an algorithm to align run lengths to the right rounds, select suitable run lengths, estimate the synthesis time* $\boldsymbol{t}$ *from the selected run lengths accurately, and finally correct substitutions of the synthesis times.*

5.4 Code construction and achievable writing rate

Based on Example 120, zero run lengths lead to deletions in alternating sequences $(\bar{\boldsymbol{a}}^{(j)})_{j=1}^{N}$ while noisy run lengths $(r_{ji})_{j=1}^{N}$ (as well as the corresponding deletions and substitutions) result in substitution errors over the synthesis time $\boldsymbol{t}_{\boldsymbol{b}}$ as well as $\boldsymbol{b}$. Therefore, our overall error-correction strategy is to protect $\boldsymbol{a}$ against deletions through deletion-correction codes and multiple sequence reconstruction algorithms and protect $\boldsymbol{b}$ by substitution-correction codes. Finally, the achievable writing rate of combining both the deletion and substitution codes is discussed at the end of Section 5.4.3.

5.4.1 Code for correcting deletions in the alternating sequence

To correct deletions in the alternating sequence $\boldsymbol{a}$, we use a trace reconstruction method based on the binary construction proposed in [1]. In this construction, each codeword is divided into blocks, each of which can correct a single deletion via the VT code. The construction can be easily extended to the q-ary alphabet by using Tenegolts q-ary single-deletion-correcting code, described

in Construction 119. An additional constraint to ensure the codewords are alternating sequences is also needed. Compared to other error-correcting codes, the VT codes and the Tq codes can achieve a low redundancy.

Construction 121. *Given the alphabet Σ_q, code length n, and block length m that divides n, the Block-based Tenegolts q-ary code (BTq code) is*

$$C_{BTq}(\boldsymbol{\alpha}, \boldsymbol{\beta}, n) = \Big\{ \boldsymbol{a} \in S_q^n \Big| \boldsymbol{a} = \boldsymbol{z}_1 \boldsymbol{z}_2 \cdots \boldsymbol{z}_b, \\ \boldsymbol{z}_i \in C_{Tq}(\alpha_i, \beta_i, m) \cap S_q^m, i \in [b] \Big\},$$

where $b = n/m$, $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \cdots, \alpha_b) \in \Sigma_q^b$, and $\boldsymbol{\beta} = (\beta_1, \beta_2, \cdots, \beta_b) \in \Sigma_m^b$.

Example 122. *Following the example in Figure 5.1, the alternating sequence $\boldsymbol{a} = ACTCGACT$ may be a codeword from $C_{BTq}(\boldsymbol{\alpha}, \boldsymbol{\beta}, n)$ over S_4^8 with $n = 8$, $b = 2$, $m = 4$, and $q = 4$. The two substrings $ACTC$ and $GACT$ are two Tq codewords from $C_{Tq}(\alpha_1, \beta_1, 4)$ and $C_{Tq}(\alpha_2, \beta_2, 4)$ respectively.*

Lemma 123. *There exist choices for $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ such that*

$$\|\mathcal{C}_{BTq}(\boldsymbol{\alpha}, \boldsymbol{\beta}, n)\| \geqslant (q-1)^n/(q^b m^b).$$

Proof. Based on Construction 121, the code $C_{BTq}(\boldsymbol{\alpha}, \boldsymbol{\beta}, n)$ is constructed over the set of alternating sequences of length n, i.e., S_q^n with $\|S_q^n\| \geqslant (q-1)^n$. Furthermore, the code $C_{BTq}(\boldsymbol{\alpha}, \boldsymbol{\beta}, n)$ is also affected by the set of parameters $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \cdots, \alpha_b) \in \Sigma_q^b$, and $\boldsymbol{\beta} = (\beta_1, \beta_2, \cdots, \beta_b) \in \Sigma_m^b$, where $\alpha_i \in [0, q-1]$ and $\beta_i \in [0, m-1]$ for $i \in [b]$. Then there exists at least one code with a set of parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ such that $\|\mathcal{C}_{BTq}(\boldsymbol{\alpha}, \boldsymbol{\beta}, n)\| \geqslant (q-1)^n/(q^b m^b)$. □

Given n, m, $\boldsymbol{\alpha}$, and $\boldsymbol{\beta}$, the set of BTq codewords can be generated by selecting alternating sequences satisfying the constraints in Construction 121. To protect an alternating sequence with length k over Σ_q, there exists a BTq code with an encoder $\mathcal{E}_{BTq} : S_q^k \to S_q^n$ such that $n = k/C_m$ asymptotically, where

$$C_m = 1 - \frac{1}{m} \log_{q-1}(qm). \tag{5.1}$$

and $q-1$ is chosen as the base of the logarithm because each position of the alternating sequence contains $q-1$ choices[2]. Based on Lemma 123, the existence of the encoder can be shown by $(q-1)^k = (q-1)^{nC_m} = (q-1)^n/(q^b m^b)$. Note that the existence of the encoder is used to explore the bound of the achievable analytical writing rate in Theorem 126. The design of such an encoder requires further exploration. The BTq codewords will be directly generated in the simulation to analyze the error probability.

[2]We assume only $q-1$ in the first round.

5.4.2 Code for correcting substitutions

This subsection presents substitution-correction codes to correct errors occurring $\boldsymbol{t_b}$ and $\boldsymbol{b}$.

As discussed earlier, the length of each run is controlled by the duration of each synthesis round. Following the precision-resolution framework [28], the synthesis time in each round is an element of the set $T_{syn} = \{t_0, t_1, \ldots, t_{L-1}\}$, with $t_b \in \mathbb{N}_+$ and $b \in \Sigma_L$, where $1 \leqslant t_0 < t_1 < \ldots < t_{L-1} \leqslant M$. Note that a larger L can increase the information per run, but also increases the probability that the synthesis time is not decoded correctly. Given the set of run lengths associated with synthesis round i, a quantizer function, discussed later, will be used to determine the index b_i of the synthesis time t_{b_i} in round i in the set $T_{syn} = \{t_0, \ldots, t_{L-1}\}$. Given the noisiness of the run lengths, substitution errors are possible in $\boldsymbol{b}$. We assume the substitution probability is bounded by $\delta/2$. Then a substitution-correcting code can be applied over $\boldsymbol{b}$ to fight against these substitution errors.

In order to explore the upper bound of the writing rate discussed in Theorem 126, we present a systematic code in the following lemma based on the Gilbert Varshamov (GV) bound.

Lemma 124. *(c.f. [63]) Given $1-1/L \geqslant \delta > 0$, there exists a systematic substitution-correcting code (SC code) over Σ_L for large n that can correct $n\delta/2$ substitution errors with asymptotic redundancy $n(1 - C_{\delta,L})$, where*

$$C_{\delta,L} = 1 + \delta \log_L \frac{\delta}{L-1} + (1-\delta)\log_L(1-\delta).$$

We refer to this code as $\mathcal{C}_{sc}$ and to its systematic encoder as $\mathcal{E}_{sc} : \Sigma_L^s \to \Sigma_L^n$, where $n = s/C_{\delta,L}$ asymptotically.

Note that the systematic code $\mathcal{C}_{sc}$ in Lemma 124 can achieve a good writing rate. In applications, a diverse set of substitution codes such as MDS codes can be applied in the PR framework based on the requirements of the writing rate and the error probability.

5.4.3 Combined codes and achievable writing rate

Given the codes discussed in the previous two subsections, we introduce the overall code for correcting both deletions and substitution errors arising in the PR framework.

Construction 125. *The BTq-SC code is defined as*

$$\mathcal{C} = \{(\boldsymbol{a}, \boldsymbol{b}) : \boldsymbol{a} \in \mathcal{C}_{BTq}, \boldsymbol{b} \in \mathcal{C}_{sc}\}.$$

Observe that $\boldsymbol{a} \in S_q^n$ and $\boldsymbol{b} \in \Sigma_L^n$.

Let $\mathcal{G}_q = (V, E, \mathcal{L})$ be the graph with $V = \Sigma_q$. For every two distinct vertices $u, v \in V$, L directed parallel edges with labels $v^{t_b}, b \in \Sigma_L$ are added from u to v, where $t_b \in T_{syn}$. Based on [28, Lemma 1], the graph $\mathcal{G}_q$ is lossless and the capacity of $S(\mathcal{G}_q)$ is

$$\mathrm{Cap}(S(\mathcal{G}_q)) = \log_2 \lambda(A_{\mathcal{G}'_q}),$$

where $\mathcal{G}'_q$ is the equivalent ordinary graph of $\mathcal{G}_q$, $A_{\mathcal{G}'_q}$ is the adjacency matrix of $\mathcal{G}'_q$, and $\lambda(A_{\mathcal{G}'_q})$ is the largest eigenvalue of $A_{\mathcal{G}'_q}$.

Based on a state-splitting encoder over the constraint graph $\mathcal{G}'_q$ and the two encoders $\mathcal{E}_{BTq}$ and $\mathcal{E}_{sc}$, the writing rate is given in the next theorem, which in contrast to [28], also takes into account the deletion-correcting code.

Theorem 126. *Given $0 < \delta \leqslant 1 - 1/L$, T_{syn}, m, and M, there exists a code $\mathcal{C}$ of Construction 125 that can achieve an asymptotic writing rate* per unit time

$$R(\mathcal{C}) \geqslant \mathrm{Cap}(S(\mathcal{G}_q))/(1 + M\alpha(1/C_{lb} - 1)),$$

where α is the sum of the probabilities of non-auxiliary vertices in the stationary distribution of the max-entropic Markov chain (MEMC) over $\mathcal{G}'_q$ [21] and $C_{lb} = \min(C_{\delta,L}, C_m)$.

Proof. Based on Construction 125, given the alternating sequence $\boldsymbol{a} \in S_q^n$ and the synthesis times in n rounds $(t_{b_1}, \cdots, t_{b_n}) \in T_{syn}^n$, every string $a_1^{t_{b_1}} \cdots a_n^{t_{b_n}}$ generated from a codeword $(\boldsymbol{a}, \boldsymbol{b})$ can be represented by a path in $\mathcal{G}_q$, belonging to the constrained system $S(\mathcal{G}_q)$. Furthermore, the capacity of the constrained system $S(\mathcal{G}_q)$ is $\mathrm{Cap}(S(\mathcal{G}_q))$.

By the state-splitting algorithm, we can build an encoder $\mathcal{E}$ for $S(\mathcal{G}_q)$ with the code rate approaching $\mathrm{Cap}(S(\mathcal{G}_q))$[28]. Given k message bits, the encoder $\mathcal{E}$ generates a sequence in $S(\mathcal{G}_q)$ with $\frac{k}{\mathrm{Cap}(S(\mathcal{G}_q))}$ synthesis time units and $\frac{k}{\mathrm{Cap}(S(\mathcal{G}_q))}\alpha$ runs asymptotically [28, Theorem 3]. Then the encoded information contains an alternating sequence and a synthesis time sequence both with length $\frac{k}{\mathrm{Cap}(S(\mathcal{G}_q))}\alpha$. By using the encoders $\mathcal{E}_{BTq}$ and $\mathcal{E}_{sc}$, the sequence with $\frac{k}{\mathrm{Cap}(S(\mathcal{G}_q))}\alpha$ runs, generated by the encoder $\mathcal{E}$, can be encoded as a codeword with at most $\frac{k}{\mathrm{Cap}(S(\mathcal{G}_q))}\alpha\frac{1}{C_{lb}}$ runs. Then the error-correcting code $\mathcal{C}$ introduces at most $\frac{k}{\mathrm{Cap}(S(\mathcal{G}_q))}\alpha(\frac{1}{C_{lb}} - 1)M$ time units for parity symbols. Since $\mathcal{C}_{sc}$ is a systematic code, the total synthesis time is at most $\frac{k}{\mathrm{Cap}(S(\mathcal{G}_q))}(1 + M\alpha(\frac{1}{C_{lb}} - 1))$ time units. Therefore, the writing rate satisfies $R(\mathcal{C}) \geqslant \mathrm{Cap}(S(\mathcal{G}_q))/(1 + M\alpha(\frac{1}{C_{lb}} - 1))$ bits per unit time. □

Figure 5.3 and Figure 5.11 show the lower bound of $R(\mathcal{C})$ given in Theorem 126. The parameters are discussed in detail in Section 5.6.1. It can be observed that rates larger than $\log_2(3)$ bits per unit time are achievable.

5.5 Decoding

In this section, we describe the decoding procedure by the combined error-correcting codes. Our goal is to decode $(\boldsymbol{a}, \boldsymbol{b})$ from $\left(\bar{\boldsymbol{a}}^{(1)}, \bar{\boldsymbol{a}}^{(2)}, \ldots, \bar{\boldsymbol{a}}^{(N)}\right)$ and $\{\bar{\boldsymbol{r}}_j\}_1^N$. First, for a given set of alternating sequences $\{\bar{\boldsymbol{a}}^{(j)}\}_1^N$, we explore an achievable error probability and compare multiple sequence reconstruction algorithms to decode $\boldsymbol{a}$ by the deletion-correcting code $\mathcal{C}_{BTq}$. Second, suppose decoding $\boldsymbol{a}$ succeeds, an alignment algorithm is proposed to identify $(r_{ji})_{j=1}^N$ for each i by using $\boldsymbol{a}$. As we will see, there is no guarantee that $(r_{ji})_{j=1}^N$ can be fully determined. Finally, a quantizer function is designed to determine b_i based on the available values of $(r_{ji})_{j=1}^N$, followed by a discussion of its error probability and the correction of substitutions over $\boldsymbol{b}$ by the substitution code $\mathcal{C}_{sc}$.

5.5.1 Decoding the alternating sequence $\boldsymbol{a}$

This subsection focuses on the decoding procedure for the alternating sequence $\boldsymbol{a}$ from $\{\bar{\boldsymbol{a}}^{(j)}\}_1^N$. We start with a discussion of the analytically achievable error probability by applying the BTq codes, followed by a comparison of multiple sequence reconstruction algorithms to decode $\boldsymbol{a}$ from $\{\bar{\boldsymbol{a}}^{(j)}\}_1^N$.

a) The achievable error probability

This subsection focuses on the analytically achievable error probability of decoding the alternating sequences, which is a reference for evaluating the performance of decoding alternating sequences.

Given an alternating sequence $\boldsymbol{a} = \boldsymbol{a}_1\boldsymbol{a}_2\cdots\boldsymbol{a}_b$ (also a BTq codeword) with b concatenated non-binary VT codewords, the enzymatic synthesis generates N alternating sequences $\{\bar{\boldsymbol{a}}^{(j)}\}_1^N$. Then the goal of this subsection is to explore the achievable error probability of decoding $\boldsymbol{a}$ from $\{\bar{\boldsymbol{a}}^{(j)}\}_1^N$, where $\bar{a}^{(j)}$ can be considered to be generated from $\boldsymbol{a}$ by suffering multiple deletions. Note that the deletion probabilities of symbols in $\boldsymbol{a}$ are affected by the L levels of synthesis times.

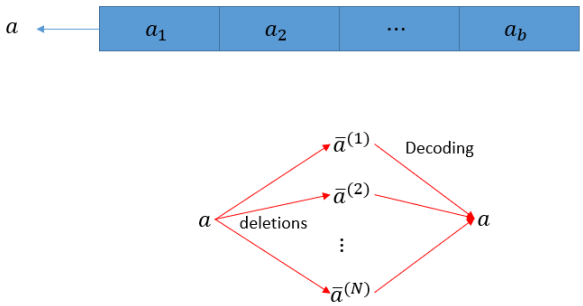


Figure 5.2: The alternating codes and the model of BTq decoding. Each codeword $\boldsymbol{a} = \boldsymbol{a}_1\boldsymbol{a}_2\cdots\boldsymbol{a}_b$ concatenates b Tq codewords. For each input $\boldsymbol{a}$, we can obtain N outputs of alternating sequences $\left(\bar{\boldsymbol{a}}^{(1)}, \bar{\boldsymbol{a}}^{(2)}, \ldots, \bar{\boldsymbol{a}}^{(N)}\right)$. The goal of the BTq decoding is to recover $\boldsymbol{a}$ from N outputs $\left(\bar{\boldsymbol{a}}^{(1)}, \bar{\boldsymbol{a}}^{(2)}, \ldots, \bar{\boldsymbol{a}}^{(N)}\right)$.

Based on the error-correction capability of BTq codes, we have the following definition.

Definition 127. *The output* $\left(\bar{\boldsymbol{a}}^{(1)}, \bar{\boldsymbol{a}}^{(2)}, \ldots, \bar{\boldsymbol{a}}^{(N)}\right), \bar{\boldsymbol{a}}^{(j)} = \bar{\boldsymbol{a}}_1^{(j)}\cdots\bar{\boldsymbol{a}}_b^{(j)}$, *for an input codeword* $\boldsymbol{a} = \boldsymbol{a}_1\boldsymbol{a}_2\cdots\boldsymbol{a}_b$, *is called* semi-alignable *if for each* $i \in [b]$, *there exists at least one* $j \in N$ *such that* $\boldsymbol{a}_i^{(j)}$ *suffers at most one deletion.*

Then the following lemma presents the achievable error probability that a BTq codeword $\boldsymbol{a}$ is not semi-alignable based on the outputs.

Lemma 128. *Let* $\boldsymbol{a} = \boldsymbol{a}_1\boldsymbol{a}_2\cdots\boldsymbol{a}_b$ *be a BTq codeword, and let* $\left(\bar{\boldsymbol{a}}^{(1)}, \bar{\boldsymbol{a}}^{(2)}, \ldots, \bar{\boldsymbol{a}}^{(N)}\right)$ *be* N *outputs, both over the alphabet* Σ_q. *Furthermore, Let* m *and* p_d *represent the block length of Tq codewords and the average probability of deleting a single symbol in the channel. Then a BTq codeword* $\boldsymbol{a}$ *is not semi-alignable with the probability*

$$P = 1 - (1 - P_e^N)^b,$$

where P_e represents the probability that the ith block $\boldsymbol{a}_i^{(j)}$ is not semi-alignable for $i \in [N]$, roughly,

$$P_e \simeq 1 - \left(\binom{m}{0} p_d^0 (1-p_d)^m + (1-\frac{1}{q}) \binom{m}{1} p_d^1 (1-p_d)^{m-1} \right).$$

Proof. Based on the analysis above, a BTq codeword $\boldsymbol{a} = \boldsymbol{a}_1 \boldsymbol{a}_2 \cdots \boldsymbol{a}_b$ is semi-alignable if and only if each concatenated Tq codeword $\boldsymbol{a}_i$ for $i \in [b]$ is semi-alignable. Let P_{ji} represent the probability that the ith Tq codeword is not semi-alignable in the jth trace by using $\boldsymbol{a}_i^{(j)}$. Without the alignment among the N independence outputs, the probability that a codeword $\boldsymbol{a}$ is not semi-alignable is

$$P = 1 - \prod_{i=1}^{b} (1 - \prod_{j=1}^{N} P_{ji}),$$

Since each Tq code can correct one deletion [91], then P_{ji} represents the probability that $\boldsymbol{a}_i^{(j)}$ suffers at least two deletions. In the rest of the proof, we assume a perfect detection of $\boldsymbol{a}_i^{(j)}$ and an expected deletion probability p_d for each nucleotide in $\boldsymbol{a}$. Then the error probability P_{ji} is identical for different i, j, i.e., $P_e = P_{ji}$. Based on the binomial distribution, the probability P_e can be obtained by excluding two cases: i) the corresponding substring suffers no deletion and ii) the corresponding substring suffers one deletion. However, when the deletion of a symbol leads to a merging of two adjacent symbols, e.g., $ACA \to AA \to A$, the corresponding substring will suffer two deletions for q choices of symbols. When p_d is very small, the merging case should be considered when we exclude the second case ii). Therefore, the probability P_e can be represented as

$$\begin{aligned} P_e \simeq & 1 - \left(\binom{m}{0} p_d^0 (1-p_d)^m + \binom{m}{1} p_d^1 (1-p_d)^{m-1} - \frac{q}{q^2} \binom{m}{1} p_d^1 (1-p_d)^{m-1} \right) \\ \simeq & 1 - \left(\binom{m}{0} p_d^0 (1-p_d)^m + (1-\frac{1}{q}) \binom{m}{1} p_d^1 (1-p_d)^{m-1} \right), \end{aligned}$$

where $\binom{m}{0} p_d^0 (1-p_d)^m$ represent case i) and $(1-\frac{1}{q})\binom{m}{1} p_d^1 (1-p_d)^{m-1}$ represents case ii) (in which the merging cases are considered). □

b) Multiple sequence reconstruction algorithms

This subsection focuses on applying multiple reconstruction algorithms to recover the BTq codeword $\boldsymbol{a}$ from the set of N outputs $\{\bar{\boldsymbol{a}}^{(j)}\}_{j=1}^{N}$, and then compares their performance with the achievable error probability in Lemma 128. Note that the BCJR algorithm with "drifts" [41], [78] requires a large size, i.e., roughly for $(q-1)^m$, of candidates for state variables for large m. Therefore, we leave it in the future and discuss the following 4 algorithms.

i) MSA-based algorithm: The first algorithm to recover $\boldsymbol{a}$ from N alternating traces $\{\bar{\boldsymbol{a}}^{(j)}\}_{j=1}^{N}$ is to apply the multiple sequence alignment (MSA) algorithm over N DNA traces followed by the majority voting algorithm, simply called the *MSA-based* algorithm. Suppose the sequence $\boldsymbol{a}'$ is obtained after a majority voting. Then we further screen $\boldsymbol{a}'$ and remove *empty-space* symbol. Compared to the following algorithm, the MSA-based algorithm suffers higher error probability

because the syndrome information of the Tq codes is not considered.

ii) Coded-based algorithm [1]: The second algorithm decodes the alternating sequence $\boldsymbol{a}$ from $\{\bar{\boldsymbol{a}}^{(j)}\}_{j=1}^{N}$ by applying a coded trace reconstruction algorithm from [1], simply called *coded-based algorithm*. Given that the code $\mathcal{C}_{BTq}$ is an extension of the binary-coded trace reconstruction method, the decoding method is similar and is described briefly here. Recall that each codeword consists of blocks capable of correcting a single deletion. In the first phase, the aim is to identify blocks that are deletion-free in at least one trace and to correct these blocks in all traces using their error-free copies. Assuming the first phase has succeeded, the remaining blocks have errors in all their copies. In the second phase, a decoder for Tenegolts code is used to correct deletion errors in these blocks to the extent possible. Note that the error probability of the coded trace reconstruction algorithm from [1] is roughly low bounded by the probability in Lemma 128.

iii) MSA-coded-based algorithm: This part *proposes* the decoding algorithm called *MSA-coded-based algorithm* consisting of the MSA-based algorithm and the Coded-based algorithm. More specifically, the decoder first decodes a string based on the MSA-based algorithm. If the string is decoded incorrectly and has a length at most n, it will be considered as a trace of the coded-based algorithm. Then the coded-based algorithm will decode the codeword by using $N+1$ traces. The motivation is that the trace recovered by the MSA-based decoding algorithm will suffer fewer errors by synchronization compared to other traces, which will benefit the decoding process. Furthermore, one more trace will also decrease the decoding error probability of the coded-based algorithm.

iv) The combined algorithm: Since the previous three decoding algorithms may function differently for different BTq codewords, this part proposes a combined algorithm consisting of three decoding algorithms. The combined algorithm starts with the MSA-based algorithm, followed by the coded-based algorithm, and the MSA-coded-based algorithms. The decoding is considered to be successful if at least one of three algorithms recover the BTq codewords.

By setting suitable parameters, Section 5.6.3 discussed the empirical error probabilities of 4 sequence reconstruction algorithms and them compared to the achievable error probability in Lemma 128.

5.5.2 Decoding the run length sequence $\boldsymbol{b}$

This subsection presents the framework of decoding synthesis times $\boldsymbol{t_b}$ and the indices $\boldsymbol{b}$ from the run lengths, including a synchronization algorithm and a quantizer function. In the subsection, we assume that $\boldsymbol{a}$ has been decoded correctly. If this is not the case, a decoding failure (possibly undetected) has occurred.

For each synthesis round i, if $(r_{1i}, \ldots, r_{Ni})$ were known, we could decode b_i using the fact that $r_{ji} \sim D(t_{b_i})$. However, our decoding is based on $\bar{\boldsymbol{a}}^{(j)}$ and $\bar{\boldsymbol{r}}_j$. As Example 120 shows, due to merging, some values in $\boldsymbol{r}_j$ are summed and appear as a single value in $\bar{\boldsymbol{r}}_j$. So to decode b_i, we first find a subset R_i of the samples $\{r_{1i}, \ldots, r_{Ni}\}$ that are known to be generated in the ith round (are not the sum of two or more elements of $\boldsymbol{r}_j$). Given this set, then a maximum-likelihood (ML) or Maximum a posteriori (MAP) quantizer can be used to decode b_i. In the following, we first show how to find R_i, then discuss the quantizer in detail.

The elements of the set R_i are determined by aligning $\bar{\boldsymbol{a}}^{(j)}$ with $\boldsymbol{a}$. We consider two specific alignments, the *forward alignment* and the *backward alignment*. In the forward alignment, starting from the left, each element α of $\bar{\boldsymbol{a}}^{(j)}$ is aligned with the leftmost non-aligned instance of α in $\boldsymbol{a}$. The backward alignment is similar but it starts from the right side and each symbol α in $\bar{\boldsymbol{a}}^{(j)}$ is mapped with the rightmost non-aligned instance of α in $\boldsymbol{a}$. Let f_{jk} be the position in $\boldsymbol{a}$ to which the kth element $\bar{r}_{jk}$ of $\bar{\boldsymbol{r}}_j$ is aligned to in the forward alignment, and let g_{jk} represent the corresponding position in the backward alignment.

Example 129. *Continuing Example 120, the forward and backward alignments of* $\bar{\boldsymbol{a}}^{(2)} = ACGACT$ *with* $\boldsymbol{a} = ACTCGACT$ *are given as*

$$\begin{array}{rcl} \boldsymbol{a} & = & ACTCGACT \\ \bar{\boldsymbol{a}}^{(2)} & = & AC \quad GACT \end{array}, \qquad \begin{array}{rcl} \boldsymbol{a} & = & ACTCGACT \\ \bar{\boldsymbol{a}}^{(2)} & = & A \quad CGACT \end{array}$$

We have $f_{21} = g_{21} = 1$, $f_{22} = 2$ *but* $g_{22} = 4$, $f_{23} = g_{23} = 5$, $f_{24} = g_{24} = 6$, $f_{25} = g_{25} = 7$, *and* $f_{26} = g_{26} = 8$. *For* $\bar{\boldsymbol{a}}_1 = ACTCACT$, *both alignment methods lead to the same alignment,*

$$\begin{array}{rcl} \boldsymbol{a} & = & ACTCGACT \\ \bar{\boldsymbol{a}}^{(1)} & = & ACTC \quad ACT \end{array}.$$

In this case, we have $f_{1k} = g_{1k} = k$ *for* $k \in [1, 4]$ *and* $f_{1k} = g_{1k} = k + 1$ *for* $k \in [6, 8]$.

The following lemma, whose proof is omitted due to space limitation, describes the relationship between $\boldsymbol{r}$ and $\bar{\boldsymbol{r}}_j$.

Lemma 130. *Suppose the* k*th element of* $\bar{\boldsymbol{a}}^{(j)}$ *is* α. *If* $f_{jk} = g_{jk} = i$, *then* $a_i = \bar{a}_{jk} = \alpha$ *and the run length of the* k*th element of* $\bar{\boldsymbol{a}}^{(j)}$ *is* $\bar{r}_{jk} = r_{jf_{jk}}$.

Proof. Suppose $a_{ji}^{r_{ji}}$ is generated in the ith round. Because of the effect of deletions, it may appear as $\bar{\boldsymbol{a}}_{jk}$ and $\bar{\boldsymbol{r}}_{jk}$ in $\bar{\boldsymbol{a}}^{j}$ and $\bar{\boldsymbol{r}}_j$, respectively. Then $f_{jk} \leqslant i$ and $g_{jk} \geqslant i$. Therefore, if $f_{jk} = g_{jk} = i$, we have $f_{jk} = g_{jk} = i$. □

When considering $\boldsymbol{x}_j$, it follows from the lemma that if the $f_{jk} = g_{jk}$, then $\bar{r}_{jk} = r_{jf_{jk}}$. For the sake of simplicity, we only use these values for the purpose of decoding. Specifically, let R_i be the multiset of values $\bar{r}_{jk}$ with $f_{jk} = g_{jk} = i$ for $j \in [N]$.

Example 131. *Continuing Example 120, we have* $R_1 = \{3, 2, 3\}$, $R_2 = \{2, 3\}$, $R_3 = \{1, 1\}$, $R_4 = \{1, 2\}$, $R_5 = \{1, 2\}$, $R_6 = \{2, 3, 3\}$, $R_7 = \{2, 1\}$, $R_8 = \{2, 1, 2\}$.

According Lemma 130, we obtain n sets of run lengths $\{R_i, i \in [n]\}$ for n rounds. Based on the observation from Example 131, the number of run lengths varies for different sets. Therefore, the rest of this subsection presents a robust estimator that can estimate the synthesis times and corrects substitutions.

a) Quantizer function $\mathcal{Q}(R_i)$

As stated above, each b_i is decoded based on R_i, which consists of run lengths $r \sim D(t_{b_i})$. Let $\hat{N}_i$ denote the number of elements in R_i. For simplicity of notation and without loss of generality, we assume $R_i = \{r_{1,i}, r_{2,i}, \ldots, r_{\hat{N}_i,i}\}$.

We assume that D is a *Poisson distribution* with parameter λt, where λ determines the expected number of nucleotides added per unit time and t is the duration of the synthesis round. Hence, for all i, j, we have $r_{ji} \sim \text{Poi}(\lambda t_{b_i})$. By rescaling the unit of time and λ, we assume t_1, the smallest synthesis time, is equal to 1. This allows us to represent the rate in a simple way, and compare with the conventional system that uses only a single synthesis time, whose rate is upper bounded by $\log_2 3$ bits. We note however that the approach is also applicable to other distributions.

The quantizer $\mathcal{Q}$ produces an estimate $\hat{b}_i$ of b_i as follows. Let $\bar{R}_i = \sum_{r \in R_i} r$ denote the sum of the elements of R_i. We define $\hat{b}_i$ as[3]

$$\hat{b}_i = \arg\max_{b \in \Sigma_L} \text{Poi}(\bar{R}_i | \lambda \hat{N}_i t_b) \pi(t_b),$$

where $\text{Poi}(k|\lambda)$ is the value of the Poisson probability mass function (pmf) with parameter λ at k and $\pi(t_b)$ is the prior probability of $t_b \in T_{syn}$. Then we estimate $\hat{b}_i$ for the ith round by the following lemma.

Lemma 132. *Given the set of synthesis times $T_{syn} = \{t_0, t_1, \cdots, t_{L-1}\}$, their prior probability $\boldsymbol{\pi} = \{\pi(t_0), \cdots, \pi(t_{L-1})\}$, and the set of run lengths in R_i following the Poisson distribution $\text{Poi}(k|\lambda t_{b_i})$, we can find $\hat{b}_i$ by comparing $\bar{R}_i$ with thresholds*

$$\rho_l = \frac{\hat{N}_i \lambda (t_l - t_{l-1}) + \ln\left(\pi(t_{l-1})/\pi(t_l)\right)}{\ln(t_l/t_{l-1})}, \quad l \in [L-1],$$

$\rho_0 = 0, \rho_L = \infty$. *Specifically, if $\rho_l \leqslant \bar{R}_i < \rho_{l+1}$, then $\hat{b}_i = l$.*

Proof. Based on the two specific alignments, a set of $\hat{N}_i$ non-zero run lengths $R_i = \{r_{1,i}, r_{2,i}, \ldots, r_{\hat{N}_i,i}\}$ are obtained for the the ith iteration. Since $r_{j,i} \sim \text{Poi}(\lambda t_{b_i})$, the output of the quantizer function is $\hat{b}_i := \mathcal{Q}(R_i)$. Let $\bar{R}_i = \sum_{r \in R_i} r$ denote the sum of the elements of R_i. Then we have $\bar{R}_i \sim \text{Poi}(\hat{N}_i \lambda t_{b_i})$. Since $\hat{N}_i$ varies for different iterations, the method in [28] cannot be applied. Based on the MAP estimation, the quantizer function can be written as

$$\begin{aligned}\hat{b}_i &= \arg\max_{b_i \in \Sigma_L} \Pr(t_{b_i} | \lambda, \bar{R}_i = k, \hat{N}_i) \\ &\propto \arg\max_{b_i \in \Sigma_L} \Pr(\bar{R}_i = k | \hat{N}_i \lambda t_{b_i}) \pi(t_{b_i}).\end{aligned}$$

where $\pi(t_{b_i}) \in \boldsymbol{\pi} = \{\pi(t_0), \cdots, \pi(t_{L-1})\}$ is the prior probability for $t_{b_i} \in T_{syn} = \{t_0, t_1, \cdots, t_{L-1}\}$.

Given $b_i \in [L]$, the key is to obtain $\hat{b}_i = b_i$ when $\bar{R}_i \in (\rho_{b_i}, \rho_{b_i+1})$, where ρ_{b_i} is the threshold between $b_i - 1$ and b_i, and ρ_{b_i+1} is the threshold between b_i and $b_i + 1$. Let us focus on the threshold

[3]We note that this estimator is not exactly the MAP estimator since the precise joint distribution $P(\boldsymbol{b}_i, R_i)$ is more complex due to the fact that the size $\hat{N}_i$ of R_i is also random and not independent of its elements.

ρ_{b_i} between $b_i - 1$ and b_i at first. Given the threshold ρ_{b_i}, the posterior probabilities satisfy

$$\begin{aligned}&\ln(\pi(t_{b_i-1})P_r(\bar{R}_i = \rho_{b_i}|\hat{N}_i\lambda t_{b_i-1})) \\ &\qquad = \ln(\pi(t_{b_i})P_r(\bar{R}_i = \rho_{b_i}|\hat{N}_i\lambda t_{b_i}))\end{aligned}$$

where $\ln(\pi(t_{b_i})P_r(\bar{R}_i = \rho_{b_i}|\hat{N}_i\lambda t_{b_i})) \propto \rho_{b_i}\ln(\hat{N}_i\lambda t_{b_i}) - \hat{N}_i\lambda t_{b_i} + \ln\pi(t_{b_i})$. Therefore, by the above equation and without considering the effect of constant, the threshold is

$$\rho_{b_i} = \frac{\hat{N}_i\lambda(t_{b_i} - t_{b_i-1}) + \ln\left(\pi(t_{b_i-1})/\pi(t_{b_i})\right)}{\ln(t_{b_i}/t_{b_i-1})}.$$

Similarly, we have the threshold ρ_{b_i+1}. Therefore, $\hat{b}_i$ is correctly estimated as b_i if $\rho_{b_i} \leqslant \bar{R}_i < \rho_{b_i+1}$. □

Based on the analysis above, the quantizer algorithm to obtain the output $\hat{b}_i := \mathcal{Q}(R_i)$ can be shown below:

- In the ith round, given $T_{syn} = \{t_0, t_1, \ldots, t_{L-1}\}$, $\boldsymbol{\pi} = \{\pi(t_0), \cdots, \pi(t_{L-1})\}$, λ, $R_i = \{r_{1,i}, \ldots, r_{\hat{N}_i,i}\}$, and $\hat{N}_i = |R_i|$, we have $\bar{R}_i = \sum_{r\in R_i} r$.
- For $1 \leqslant l \leqslant L-1$, the threshold between l and $l+1$ is computed as

  $$\rho_l = \frac{\hat{N}_i\lambda(t_{l+1} - t_l) + \ln\left(\pi(t_l)/\pi(t_{l+1})\right)}{\ln(t_{l+1}/t_l)}.$$

  Furthermore, since $0 \leqslant \bar{R}_i < \infty$, let $\rho_0 = 0$ and $\rho_L = \infty$.
- The synthesis time is quantizered as $\hat{b}_i = l$ if $\rho_l \leqslant \bar{R}_i < \rho_{l+1}$, where $l \in \{0, 1, \cdots, L-1\}$.

Based on the quantizer $\hat{b}_i := \mathcal{Q}(R_i)$, the estimation of the index $\hat{b}_i$ for the ith round may fail if the run lengths $\bar{R}_i < \rho_l$ and $\bar{R}_i \geqslant \rho_{l+1}$. Given the set of run lengths R_i and $\hat{N}_i$, the error probability of estimating $\hat{b}_i$ is the sum of probabilities with run lengths smaller than ρ_l or not smaller than ρ_{l+1}. However, due to the deletions and substitutions occurring in the run lengths, the size $\hat{N}_i$ of R_i varies for different inputs and rounds. Therefore, it is difficult to determine the probability of incorrectly decoding b_i without R_i. Instead, an approximate lower bound may be obtained by considering the probability of incorrect decoding based on all N run lengths $(r_{1i}, \ldots, r_{Ni})$. Assuming a prior probability vector $\boldsymbol{\pi} = (\pi(t_0), \ldots, \pi(t_{L-1}))$, this probability is given as

$$P_e(N, \boldsymbol{\pi}) = \sum_{l=0}^{L-1} \pi(t_l)\,\mathrm{Poi}(\bar{R}_i < \rho_l \cup \bar{R}_i \geqslant \rho_{l+1}|N\lambda t_l),$$

where $\bar{R}_i = \sum_{j=1}^{N} r_{j,i}$. Note that the error probabilities for $l = 0$ and $l = L-1$ only consider probabilities with run lengths not smaller than ρ_1 and smaller ρ_{L-1} respectively since $\rho_0 = 0$ and $\rho_L = \infty$.

For information symbols of the codewords $\boldsymbol{b} \in \mathcal{C}_{sc}$, the prior distributions $\boldsymbol{\pi}$ can be obtained from the MEMC over $\mathcal{G}_q'$, which we denote by $\boldsymbol{\pi}_M$. To approximate the average probability of error

for all symbols of the codeword, we assume that the parity symbols have a uniform distribution $\boldsymbol{\pi}_U$. Then the lower bound on the probability of symbol error can be approximated as

$$P_e(N) \simeq C_{\delta,L} P_e(N, \boldsymbol{\pi}_M) + (1 - C_{\delta,L}) P_e(N, \boldsymbol{\pi}_U), \tag{5.2}$$

where $P_e(N)$ is derived by applying a systematic substitution code approaching the GV bound of code rate. We may derive the expected error probability if N is replaced by the expected number of traces.

b) Correcting substitutions

By executing the quantizer function for n rounds, we will recover $\hat{\boldsymbol{b}}$, where each symbol in the systematic code C_{sc} may suffer a substitution with the error probability lower bounded by $P_e(N)$. If $d_H(\hat{\boldsymbol{b}}, \boldsymbol{b}) \leqslant \delta n/2$, then $\boldsymbol{b}$ can be identified correctly. By choosing suitable δ, we can achieve a tradeoff between the writing rate and the capability of correcting substitution errors.

5.6 Code parameters and simulation results

In this section, we will first briefly discuss some of the code parameters and then present the numerical results.

5.6.1 General parameters

In the simulation, we let $L = 3$, $q = 4$, $\boldsymbol{a} \in S_q^n$, $\boldsymbol{b} \in \Sigma_L^n$, $T_{syn} = \{t_0, t_1, t_2\} = \{1, 2, 3\}$, $M = t_2 = 3$. Based on the MEMC over $\mathcal{G}'_q$, the prior probabilities for elements in $T_{syn} = \{1, 2, 3\}$ are $\boldsymbol{\pi}_M = (\pi(t_0), \pi(t_1), \pi(t_2)) = (0.759, 0.192, 0.049)$. Furthermore, the parameter α in Theorem 126 is $\alpha = 0.7756$. The number of traces N is in the range $\{10, 15, 20, 25, 30\}$. Furthermore, each BTq codeword is generated by concatenating $b = 4$ Tq codewords, each with length m.

We note that given some aspects of our code construction are not explicit, they are not fully implemented in the simulation but rather the results are obtained based on their properties. In particular, the code $\mathcal{C}_{sc}$ is not explicit. We generate a set of strings $\boldsymbol{b}$ to simulate codewords from $\mathcal{C}_{sc}$ with the writing rate $C_{\delta,L}$. For each string $\boldsymbol{b} \in \Sigma_L^n$, the first $\lfloor nC_{\delta,L} \rfloor$ message symbols satisfy the distribution $\boldsymbol{\pi}_M$ and the other $\lceil n(1 - C_{\delta,L}) \rceil$ parity symbols satisfy the uniform distribution $\boldsymbol{\pi}_U$. For $\boldsymbol{a}$, a random codeword from the code $\mathcal{C}_{BTq}$ with $\boldsymbol{\alpha} = (0, 0, 0, 0), \boldsymbol{\beta} = (1, 2, 3, 4)$ is chosen. Traces are generated based on codewords $(\boldsymbol{a}, \boldsymbol{b})$ and then decoded to produce $(\hat{\boldsymbol{a}}, \hat{\boldsymbol{b}})$. If $\boldsymbol{a} = \hat{\boldsymbol{a}}$ and $d_H(\boldsymbol{b}, \hat{\boldsymbol{b}}) \leqslant \lfloor n\delta/2 \rfloor$, decoding has succeeded. Unless otherwise stated, 5×10^5 codewords $(\boldsymbol{a}, \boldsymbol{b})$ are stored in DNA in the simulation.

5.6.2 Writing rate

This subsection presents that our construction BTq-SC code can achieve a writing rate higher than $\log_2 3$ bits per unit time. For simplicity, the block length is fixed to be $m = 56$, then the length

of each BTq codeword is $n = 4m = 224$. To analyze the effect of the parameter δ, we set δ in the range $\{0.016, 0.018, 0.020, 0.022\}$.

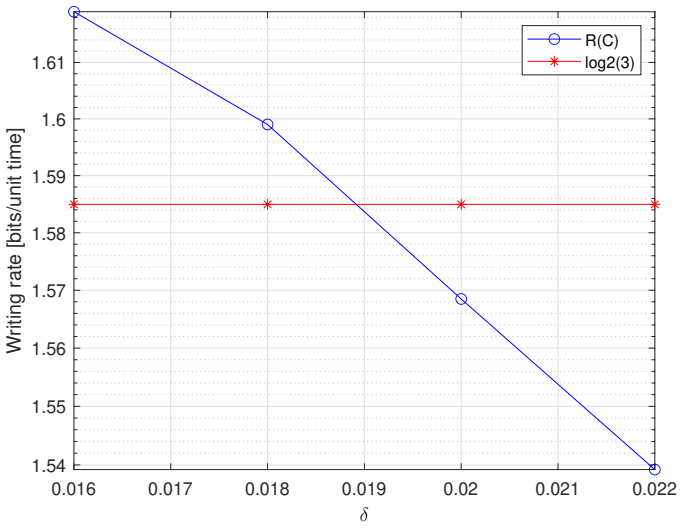


Figure 5.3: The lower bound of the writing rate $R(\mathcal{C})$ of BTq-SC codes with respect to $\delta \in \{0.016, 0.018, 0.020, 0.022\}$. Here $m = 56$ and $L = 3$.

Based on Theorem 126, Figure 5.3 shows a lower bound of $R(\mathcal{C})$ with respect to $\delta \in \{0.016, 0.018, 0.020, 0.022\}$. The curve with stars denotes the upper bound of writing rate, $\log_2 3$ bits per unit time, of codes in [38], and the curve with circles represents the lower bound of $R(\mathcal{C})$ in Theorem 126. Based on the numerical results, the BTq-SC code can achieve a higher substitution correction capability at the cost of the writing rate. More specifically, given $m = 56$, $R(\mathcal{C})$ can be larger than $\log_2(3)$ bits per unit time when $\delta \leqslant 0.018$.

5.6.3 Decoding BTq codewords

In this subsection, we compared the error probabilities of four sequence reconstruction algorithms to decode BTq codes (alternating sequences). Note that the BTq codewords are generated by concatenating b Tq codewords, where each Tq codeword is generated via a syndrome check of a length-m alternating sequence. A systematic encoding/decoding algorithm will be left for future work.

Based on Lemma 128, the error probability of BTq codewods is affected by the expected deletion probability p_d, the block length m, and the number of traces. If the number of deletions in each block of length m is too large, the error-correcting capabilities of the BTq code will become ineffective. Based on Poisson distribution, the deletion probability of a given run is at most $e^{-\lambda}$ when $t_0 = 1$. Since there are L levels of synthesis time, the expected deletion probability for Lemma 128 is approximately

$$p_d = C_{\delta,L} \sum_{l=0}^{L-1} \pi(t_l) e^{-\lambda t_l} + \frac{(1 - C_{\delta,L})}{L} \sum_{l=0}^{L-1} e^{-\lambda t_l}. \tag{5.3}$$

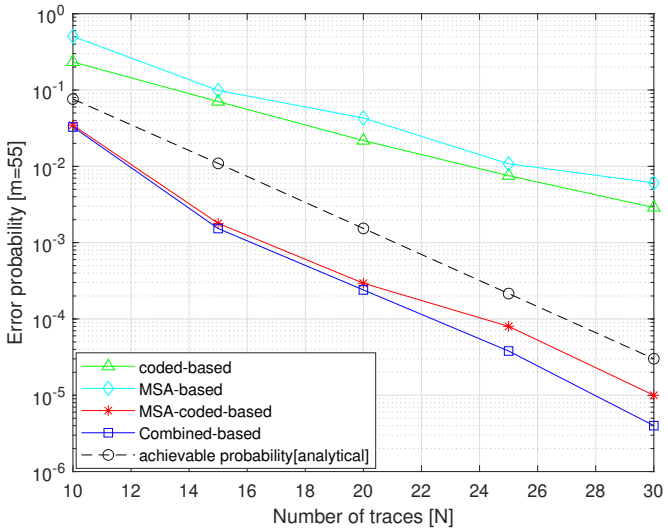


Figure 5.4: The error probability of decoding BTq codewords with respect to the number of traces. Here, we let $\lambda = 3.0$, $m = 56$, and $\delta = 0.018$. The length of BTq codewords is $n = bm = 224$.

Hence the expected number of deleted runs in each block is at least mp_d (note that more runs can be deleted due to merging). Based on Figure 5.3, we let $m = 56$, $\delta = 0.018$, and $\lambda = 3$ to achieve a writing rate higher than $\log_2 3$ bits per unit time[4]. Each BTq codeword consists of 220 rounds.

Figure 5.4 compares the decoding error probability of the four decoding algorithms in Subsection 5.5.1 with respect to the number of DNA strands $N \in \{10, 15, 20, 25, 30\}$. More specifically, the error probabilities of the coded-based algorithm, the MSA-based algorithms, the MSA-coded-based algorithm, and the combined algorithms are represented by curves with the upper arrows, diamonds, stars, and squares respectively. Since the number of deleted runs $\xi = mp_d \simeq 2$ is higher than one run which can be corrected by a Tq code, both the coded-based algorithm and the MSA-based algorithm suffer high error probabilities. However, by combining the MSA-based algorithm with the coded-based algorithm, the MSA-coded-based algorithm can significantly decreases the error probability. In other words, the output generated by the MSA algorithm may suffer at most one deletion in each Tq block. However, the effect of the MSA algorithm decreases when N is large. Finally, the numerical results showed that the combined algorithm and the MSA-coded-based algorithm outperform the other two algorithms and reach the analytical error probability in Lemma 128.

5.6.4 Error probability of Quantizer function

This subsection discusses the error probability of quantizer function $\hat{b}_i := \mathcal{Q}(R_i)$. Furthermore, following the previous discussion, we set $m = 56$, $\lambda = 3$, $\delta = 0.018$. Note that, in the rest of the simulation results, we assume that the BTq codeword $\boldsymbol{a}$ is successfully decoded by the combined algorithm.

To discuss the quantizer function, the BTq codeword $\boldsymbol{a}$ is assumed to be decoded correctly and

[4]The work [38] shows that the length of raw sequences is around 3 times the length of the compressed sequence (alternating sequences).

n sets of run lengths R_i for $i \in [n]$ have been acquired. By assuming all N run lengths available for the ith round, $P_e(N)$ in (5.2) provided an analytically lower bound of the error probability of quantization. However, the zero run lengths and merging of runs in real situations will reduce the size of R_i. Let N_a denote the average number of acquired run lengths in each of n rounds. By only considering the effects of zero run lengths and merging runs, we have

$$N_a(N) \simeq N\left(1 - (1 + \frac{2}{q})p_d\right), \tag{5.4}$$

where $\frac{2}{q}p_d$ denotes the expected number of runs affected by the merging of run lengths.

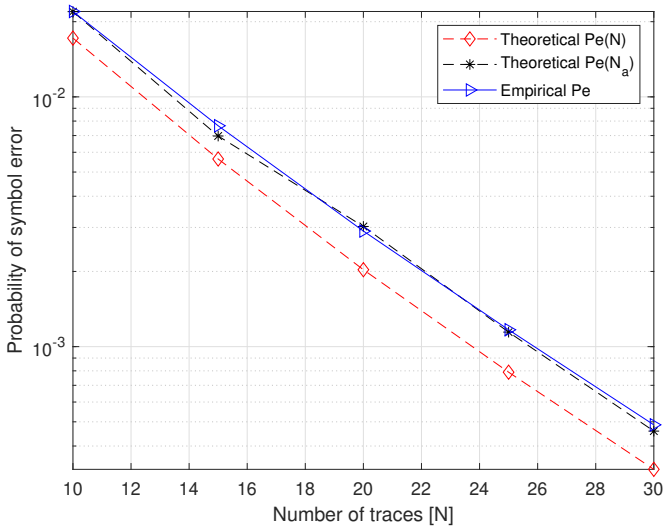


Figure 5.5: The comparison of empirical and theoretical error probabilities of estimating a synthesis time t_{b_i} (equivalently b_i) by the quantizer function. Let $\delta = 0.018$, $\lambda = 3$, and $m = 56$.

Figure 5.5 compares the analytical and empirical symbol error probabilities by the quantizer function. The two analytical errors are based on $P_e(N)$ and $P_e(N_a)$ given in (5.2) and N_a in (5.4), which assume all run lengths and an expected number of run lengths are available, respectively. It can be seen that the empirical error probability approximates the analytical error probability $P_e(N_a)$, and is also close to the lower bound $P_e(N)$, implying that the synchronization method for identifying the run lengths (in Subsection 5.5.2) is effective. Furthermore, we can observe that the error rate falls nearly exponentially as N increases.

5.6.5 Analysis of decoding b and a tradeoff

Based on figure 5.5, the empirical average error probability of the quantizer function approximates the analytical error probability $P_e(N_a)$. Furthermore, we assume that the run lengths generated in different rounds are independent. If the substitution errors of the quantizer function are also independent, the number of substitution errors in $\boldsymbol{b}$ satisfies the *Binomial distribution*,

$x \sim \text{Bin}(n, P_e(N_a))$. Then the probability mass function (pmf) of $d_H(\hat{\boldsymbol{b}}, \boldsymbol{b}) = k$ satisfies

$$\Pr(d_H(\hat{\boldsymbol{b}}, \boldsymbol{b}) = k) = \binom{n}{k} P_e(N_a)^k (1 - P_e(N_a))^{n-k}. \tag{5.5}$$

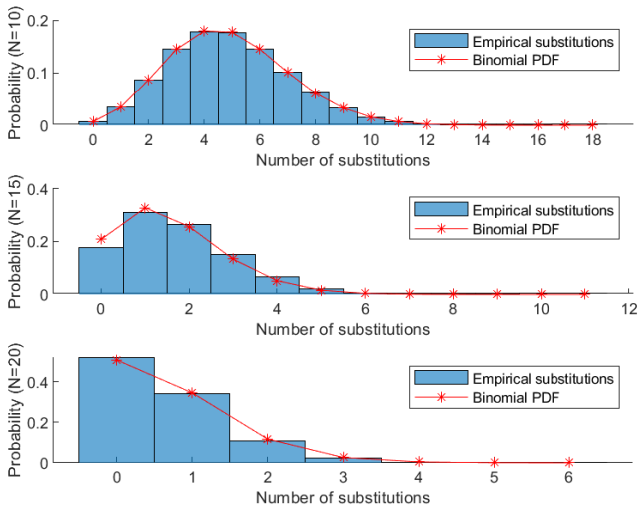


Figure 5.6: The pmf of Hamming distance $d_H(\hat{\boldsymbol{b}}, \boldsymbol{b})$ with different number of traces N at the output. In the simulation, we set $\lambda = 3.0$, $m = 56$, and $\delta = 0.018$.

Based on the assumption in (5.5), Figure 5.6 compares the probability of $d_H(\hat{\boldsymbol{b}}, \boldsymbol{b}) = k$ and $\text{Bin}(n, P_e(N_a))$ in terms of different number of traces N. More specifically, the histograms represent the probability of the substitutions, and the curves with stars denote the pmf of $\text{Bin}(n, P_e(N_a))$. For $N = 10, 15, 20$, we observe that the probabilities of $d_H(\hat{\boldsymbol{b}}, \boldsymbol{b}) = k$ approximates the pmf of the binomial distribution perfectly.

Therefore, an error-correcting code $\mathcal{C}$ can correct at most d substitutions with error probability roughly

$$P_e(\mathcal{C}, d) = 1 - \sum_{k=0}^{d} \Pr(d_H(\hat{\boldsymbol{b}}, \boldsymbol{b}) = k) \simeq 1 - \Pr(x \leqslant d), \tag{5.6}$$

where $x \sim \text{Bin}(n, P_e(N_a))$. Recall that there exists a systematic substitution-correcting code satisfying the GV bound that can correct $\lfloor n\delta/2 \rfloor = 2$ substitutions. Based on Figure 5.6, the $\mathcal{C}_{sc}$ can decode δ with low error probabilities when $N \geqslant 20$. However, it suffers a high error when N is small. Then we may apply error-correcting codes $\mathcal{C}$ with a larger d to achieve a lower error probability. However, a larger d results in a larger δ. Therefore, it is necessary to analyze the effect of δ on the probabilities of the quantizer function and $\Pr(d_H(\hat{\boldsymbol{b}}, \boldsymbol{b}) = k)$.

Figure 5.7 shows the effects of δ on the error probabilities of estimating synthesis times by the quantizer function. The two curves with stars represent the error probabilities acquired by letting $\delta = 0.018$, and the two curves with circles are acquired by letting $\delta = 0.05$. Based on the numerical results, for a specific $\delta \in \{0.018, 0.05\}$, the empirical error probability represented by solid curves approximates the analytical error probability. Furthermore, by increasing δ from 0.018 to 0.05,

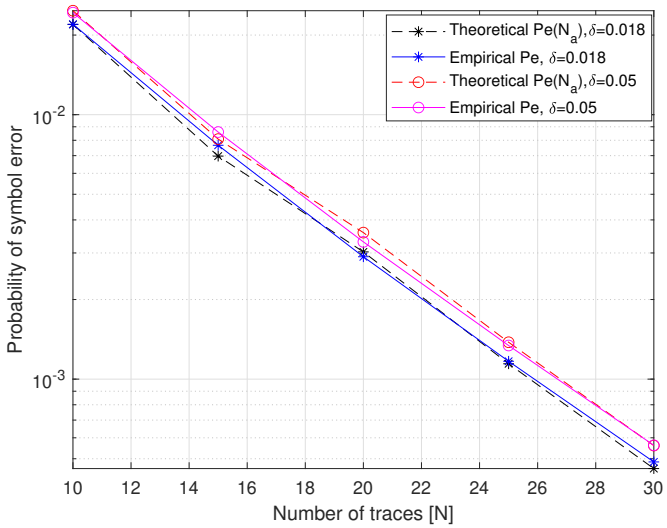


Figure 5.7: The effects of δ on the empirical and theoretical error probabilities of estimating t_{b_i} (equivalently b_i). Let $\delta \in \{0.018, 0.05\}$, $\lambda = 3$, and $m = 56$.

both the empirical and analytical error probabilities have a limited increasing. Therefore, slightly increasing δ in a small range does not obviously affect the error probability of the quantizer function as well as the pmf of $\Pr(d_H(\hat{\boldsymbol{b}}, \boldsymbol{b}) = k)$.

Since $n = 224$ is large enough, there exists a systematic substitution-correcting code $\mathcal{C}_{sc}$ that can correct $d = \lfloor n\delta/2 \rfloor = 5$ substitutions for a larger $\delta = 0.05$. Based on the analysis, given a large n, by increasing δ, we can select a suitable code $\mathcal{C}$ capable of correcting more substitutions, which achieves a tradeoff between the writing rate and error probability.

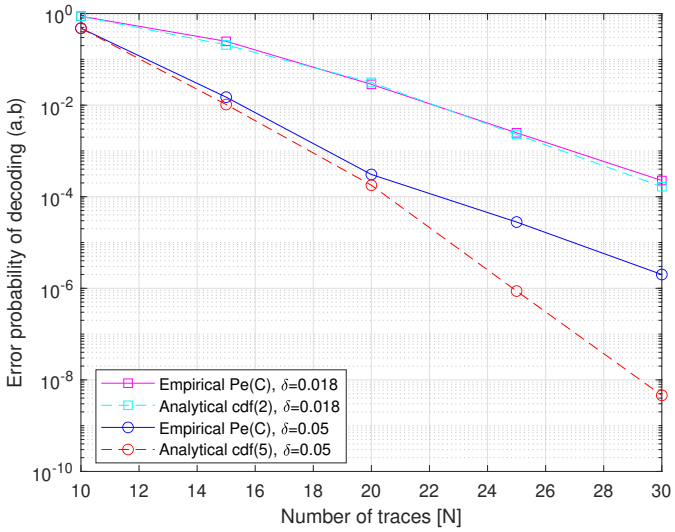


Figure 5.8: The error probability of decoding BTq-SC codewords. In this setting, let $m = 56$, $\lambda = 3$, and $\delta \in \{0.018, 0.05\}$.

Figure 5.8 analyzes the error probability of decoding BTq-SC codewords, $P_e(\mathcal{C}, d)$, with respect

to the number of DNA sequences N for $\delta \in \{0.018, 0.05\}$. In particular, the two curves with squares represent the error probability when $\delta = 0.018$, and the two curves with circles are for $\delta = 0.05$. For each specific δ, the empirical error probability approximates the probability $1 - \Pr(x \leqslant d)$ for $d = \lfloor n\delta/2 \rfloor$ and $x \sim \mathrm{Bin}(n, P_e(N_a))$. Furthermore, as N increases, the error probability of decoding BTq-SC codewords quickly diminishes. By increasing δ from 0.018 to 0.05, the empirical error probability $P_e(\mathcal{C}, d)$ decreases obviously. Furthermore, the gap increases as N increases. Therefore, by setting suitable δ, we can achieve a tradeoff between the writing rate and the error probability. However, if both δ and N are large, the error probability of decoding the substitution codewords, i.e., $\boldsymbol{b}$, will be lower bounded by the error probability of decoding BTq codewords $\boldsymbol{a}$.

Based on Figure 5.8, given a set of parameters λ, n, T_{syn}, the decoding error probability $P_e(\mathcal{C}, d)$ can be represented as

$$P_e(\mathcal{C}, \lfloor n\delta/2 \rfloor) \simeq 1 - \Pr(x \leqslant \lfloor n\delta/2 \rfloor), \tag{5.7}$$

where $x \sim \mathrm{Bin}(n, P_e(N_a))$ and $P_e(N_a)$ is also a function of δ. Therefore, without considering the writing rate, the error probability $P_e(\mathcal{C}, d)$ with respect to δ can be shown in the following curve. Then for a specific target BTq-SC decoding error probability $P_e(\mathcal{C}, d)$, we can find suitable δ as well as the $\mathcal{C}_{sc}$.

5.6.6 Effects of parameters

This subsection analyzes the effects of parameters, *i.e.*, δ, λ, m, and N, on the empirical error probabilities.

Figure 5.8 presents the error probability with respect to the parameter δ. Based on the simulation results, increasing δ from 0.018 to 0.05 will obviously decrease the error probability. However, if N is large, the benefits of a large δ will be diminished by the error probability of decoding BTq codewords, i.e., $\boldsymbol{a}$.

Figure 5.9 discussed the effect of error probability of decoding BTq codewords with respect to the block length m. As m decreases from 56 to 36, the error probability of decoding BTq codewords decreases rapidly. Furthermore, the gap between them also increases as N increases.

Figure 5.10 discusses the error probability of decoding $\boldsymbol{b}$ with respect to the expected number of nucleotides added per unit time λ. By increasing λ from 3.0 to 3.5, the error probability of decoding $\boldsymbol{b}$ also decreases rapidly. Note that under the setting of $\lambda = 3.5, m = 56$, and $\delta = 0.018$, the DNA storage with enzymatic synthesis can achieve a writing rate $\geqslant \log_2 3$ bits per unit time and an error probability as low as 10^{-5}.

Based on the analysis above, increasing λ, δ, and decreasing m can increase the error probability of decoding BTq-SC codewords. Furthermore, we can design a DNA storage system with target error probability by i) setting suitable δ, λ, and N satisfying $P_e(\mathcal{C}, d)$ and ii) decreasing m necessarily if the error probability of decoding $\boldsymbol{a}$ restricts the error probability of decoding $\boldsymbol{b}$.

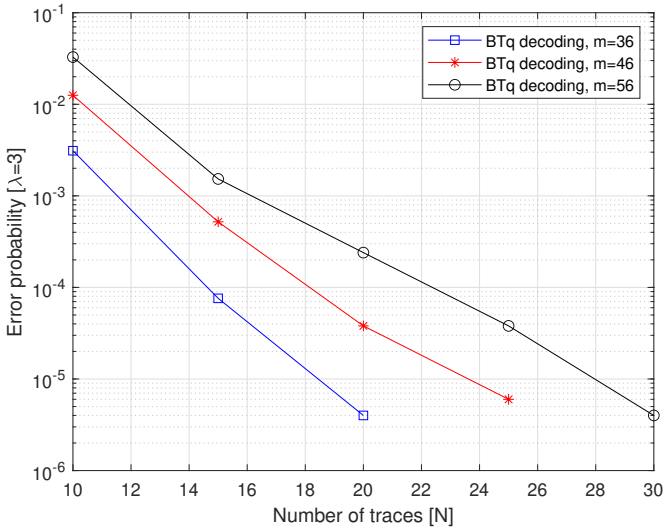


Figure 5.9: The error probability of decoding BTq codewords with respect to block length m. In this setting, let $\lambda = 3$, and $\delta = 0.018$, and $m \in [36, 46, 56]$. Based on the plot, decreasing m will decrease the error probability of decoding BTq codes.

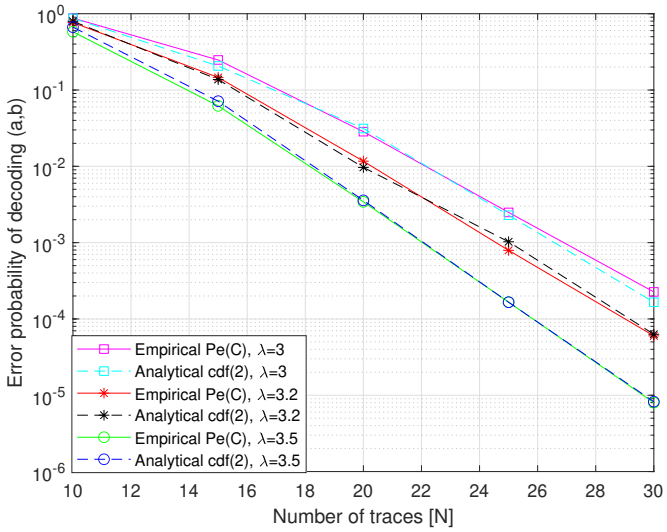


Figure 5.10: The error probability of decoding BTq-SC codewords with respect to different λ. In this setting, let $\lambda \in [3, 3.2, 3.5]$, and $\delta = 0.018$, and $m = 56$. Based on the plot, increasing λ will decrease the error probability of decoding BTq-SC codes.

5.6.7 Explicit error-correcting codes

Based on Figure 5.3 and Figure 5.10, the error-correcting code in Construction 125 can achieve a writing rate higher than $\log_2 3$ bits per unit time and an error probability approximating (5.7). However, one important question is whether an explicit substitution-error-correcting code $\mathcal{C}_{sc}$ can be applied in Construction 125. As far as we know, the Reed-Solomon (RS) codes can achieve the singleton bound of a substitution channel with a high code rate. Therefore, this subsection presents

an explicit Construction 125 called BTq-RS code, i.e., $\mathcal{C}_{BTq-RS} = \{(\boldsymbol{a}, \boldsymbol{b}), \boldsymbol{a} \in \mathcal{C}_{BTq}, \boldsymbol{b} \in \mathcal{C}_{RS}\}$.

To simplify the simulation, we reset $L = 2$ since the RS codes in matlab is over binary. Let $q = 4$, $\boldsymbol{a} \in S_q^n$, $\boldsymbol{b} \in \Sigma_L^n$, $T_{syn} = \{t_0, t_1\} = \{1, 2\}$, $M = t_2 = 2$. Based on the MEMC over $\mathcal{G}_q'$, the prior probabilities for elements in $T_{syn} = \{1, 2\}$ are $\boldsymbol{\pi}_M = (\pi(t_0), \pi(t_1)) = (0.7913, 0.2087)$. Furthermore, the parameter α in Theorem 126 is $\alpha = 0.8273$. The number of traces N is in the range $\{10, 15, 20, 25, 30\}$. Furthermore, each BTq codeword is generated by concatenating $b = 4$ Tq codewords, each with length $m = 57$. Then the length of codewords is $n = bm = 228$. Note that 2×10^5 codewords $(\boldsymbol{a}, \boldsymbol{b})$ are generated in this simulation.

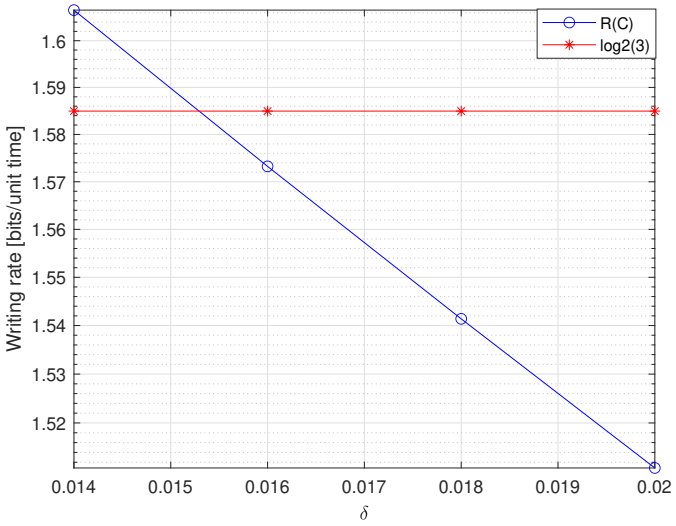


Figure 5.11: The lower bound of the writing rate $R(\mathcal{C})$ of BTq-SC codes with respect to $\delta \in \{0.014, 0.016, 0.018, 0.020\}$. Here $m = 57$ and $L = 2$.

Based on Theorem 126, Figure 5.11 shows a lower bound of the writing rate $R(\mathcal{C})$ with respect to $\delta \in \{0.014, 0.016, 0.018, 0.020\}$. Based on the numerical results, the BTq-SC code can achieve a writing rate $R(\mathcal{C})$ larger than $\log_2(3)$ bits per unit time when $\delta \leqslant 0.015$. Therefore, in the rest of the simulation, we let $\delta = 0.015$ to guarantee that the RS code can correct more substitutions.

Given $n = 228$, $\delta = 0.015$, and $L = 2$, the systematic substitution code $\mathcal{C}_{sc}$ at most can hold $n(1 - C_{\delta,L}) = 26$ redundant symbols. In order to achieve a writing rate higher than $\log_2 3$ bits per unit and high error-correcting capability, we construct the Reed-Solomon code $RS(38, 34, 5)_{2^6}$ as the systematic substitution-error-correcting code, where each codeword contains $4 \times 6 = 24$ parity symbols and can correct 2 substitutions. Since the SC code is required to correct $\lfloor n\delta/2 \rfloor = 1$ symbol, the BTq-RS error-correcting code $\mathcal{C}_{BTq-RS}$ is a valid code of the Construction 125.

Figure 5.12 presents the error probability of decoding BTq-RS codewords with respect to the number of traces. The solid curve represents the empirical error probability of decoding BTq-RS codewords and the dashed curves represent the analytical error probability of correcting $\lfloor n\delta/2 \rfloor = 1$ or 2 substitution errors. Based on the numerical results, the BTq-RS code can achieve an empirical error probability approximating the analytical error probability $P_e(\mathcal{C}, d)$ (5.6) with $d = 2$, where $x \sim \text{Bin}(n, P_e(N_a))$ and $P_e(N_a)$ is derived by (5.2),(5.4), and (5.3) with $L = 2$. Therefore, we

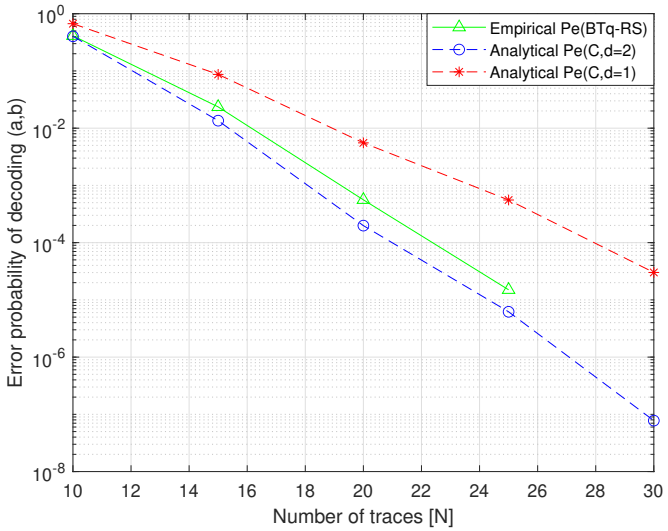


Figure 5.12: The error probability of decoding BTq-RS codewords with respect to the number of traces. In this setting, let $L = 2$, $\lambda = 3.2$, $\delta = 0.015$, and $m = 57$.

have an accurate analytical symbol error probability of the quantizer function (5.2). Furthermore, by choosing suitable δ, the valid construction BTq-RS code can achieve tradeoff between the error probability and the writing rate.

5.7 Summary and limitations

By a new inexpensive enzymatic synthesis method [38], the current chapter proposed an error-correcting code and decoding algorithms to correct deletions (of runs), the dominant errors, over DNA traces and achieve a writing rate higher than $\log_2 3$ bits per unit time. Based on the *precision-resolution* (PR) framework [67], this section proposed error-correcting codes called *BTq-SC code*, consisting of a *block-based Tenegolts q-ary* (BTq) code [1] over alternating strings and a *substitution-correcting* (SC) code over synthesis times, and a decoding algorithm consisting of sequence reconstruction algorithms, synchronization algorithms, and Bayes estimation. Compared to the codes either with a writing rate lower than $\log_2 3$ bits per run in [38] or not being able to correct deletions in [28], the proposed code can achieve a writing rate higher than $\log_2 3$ bits per unit time and a decoding error probability approximating the analytical error probability. By modifying different parameters, we can achieve tradeoff between a (low) target error probability and the writing rate.

As the code construction and the decoding algorithm perform good, one open problem is to design an encoder with the code rate approximating (5.1) for the BTq code over alternating sequences. For example, the recently proposed systematic encoder in [58] can be a good reference for our problem.

Acknowledgement

The research presented in this chapter benefited from the helpful discussions and insightful comments from Dr. Mete Civelek.

Chapter 6

Conclusion and open problems

Due to the advantages of high density, longevity, and ease of generating copies, DNA is considered a promising alternative for future data storage. However, a diverse set of errors will occur in synthesis, storage, and sequencing of DNA storage. In this dissertation, we studied the problems of correcting multiple sources of errors over DNA data storage, i.e., i) fixed-length (tandem) duplications (k-TDs) and one substitution, ii) short tandem duplications and at most p edits, iii) a substring edit error, and iv) noisy run lengths and deletions (of runs) from enzymatic synthesis. More specifically, we proposed error-correcting codes and decoding algorithms that can correct more errors at a low extra cost compared to previous works in different settings. The rest of the chapter first concludes the main contributions of this chapter, followed by the discussion of future open problems.

6.1 Conclusion

Chapter 2 studied the problem of constructing error-control codes to fight against an unlimited number of fixed-length duplications and at most one substitution. We focused on two noise models, where the substitution error is either restricted to one duplication event (called *noisy duplication* or *restricted substitution*), or may occur at any position in the string (called *unrestricted substitution*). First, we designed an error-detecting code in Construction 17 and Theorem 18 to detect another restricted substitution (apart from an unlimited number of k-TDs) at an extra cost of $O(\log k)$ bits. Second, Construction 26 presented error-detecting codes for any k-TDs and at most one unrestricted substitution, respectively. For a noisy duplication channel, we constructed error-correcting codes in Construction 37 to correct a restricted substitution with an extra redundancy of roughly $(2k + 4) \log_q n$ symbols, without extra loss in asymptotic rate loss.

Chapter 3 dealt with the problem of correcting an unlimited number of short duplications (with length upper bounded by 3) and at most p edits. To achieve this goal, we start with the problem of correcting one extra edit error. Hence, we first constructed (Reed-Solomon-based) error-correcting codes in Construction 52 to correct any number of short tandem duplications and at most one edit at an extra cost of 0.003 bits/symbol compared to the asymptotically optical codes for short duplications only. After that, we proposed new error-correcting codes in Construction 68 to correct an unlimited number of short duplications and at most p substitutions based on the

idea of concatenating blocks into groups. However, the code construction suffers an obvious rate loss asymptotically [86]. Therefore, to further reduce the redundancy, we constructed an error-correcting codes in Construction 90 to correct an arbitrary number of short tandem duplications and at most p edits by applying the syndrome compression technique [75] and Construction 90 to protect the syndrome redundant information. When $q \geqslant 4$ and p are constant, the state-of-the-art error-correcting codes in Construction 90 achieves the same asymptotic code rate as the error-correcting codes for $\leqslant$3-TDs only [30], with an extra redundancy of roughly $8p \log_q n$ symbols, and has polynomial time complexities in the encoding and decoding processes.

Chapter 4 focused on correcting a k-substring edit as it will have many localized errors such as a burst of insertions/deletions/substitutions as a special case. First of all, we presented the motivation of correcting substring edits by the statistical hypothesis tests in Table 4.2 and the codes reaching the GV bounds in Figure 4.5 and Figure 4.6, showing that the substring edits are common errors and substring-edit-correcting codes can achieve a lower redundancy compared to insertion/deletion-error-correcting codes. Then this chapter constructed error-correcting codes in Construction 117 to correct a k-substring edit with the redundancy of roughly $2 \log n$ and polynomial time complexities in both encoding and the decoding algorithms.

Because of the promising future of the inexpensive enzymatic synthesis, Chapter 5 proposed an error-correcting code and a decoding algorithm to fight against noisy run lengths and deletions (of runs) over DNA storage and achieve a writing rate higher than $\log_2 3$ bits per unit time. Based on the *precision-resolution* (PR) framework [67], we first presented an error-correcting codes in Construction 125 called *BTq-SC code*, consisting of a *block-based Tenegolts q-ary* (BTq) code [1] over alternating strings and a *substitution-correcting* (SC) code over synthesis times, followed by a decoding algorithm including sequence reconstruction, statistical estimation, and the analysis of symbol error probability. Finally, we discussed the strategies for modifying parameters that can achieve a tradeoff among target error probability, data rate, and other requirements.

6.2 Open problems

The first possibly challenging problem is correcting multiple edits and duplications of length bounded by an arbitrary constant k. If k is larger than 3, the duplication root is no longer unique [30], which complicates the code design. Furthermore, a key feature of duplications of length at most 3 is that such duplications lead to regular languages. We used this fact to characterize the effect of the channel on the roots of sequences. However, if $k \geqslant 4$, then the language is not regular [46], leading to challenges in characterizing the channel, especially under the effect of multiple edits. To guarantee a unique root under $\leqslant k$ duplications with $k \geqslant 4$, the work [8] constructed error-correcting codes by selecting a set of $(k+1)$-distinct strings with $q \geqslant (k+1)$ and achieved an asymptotic code rate of $\frac{1}{k}\sum_{i=1}^{k} \log(q-i)$. This means that, despite the interesting approach, to correct duplications of length $\geqslant 4$, [8] requires $q \geqslant 5$, which is not suitable for the important application of data storage in DNA with $q = 4$.

As analyzed in Section 4, we constructed error-correcting codes to correct a substring edit with

redundancy roughly $2\log n$, which is higher than the lower bound of roughly $\log n$ derived from the special case of correcting a burst of deletions [66]. Therefore, one important open problem is to construct error-correcting codes with redundancy of roughly $\log n$ bits. Furthermore, another important open problem is to extend the current method to correct multiple substring edit errors.

For the DNA storage model with enzymatic synthesis, Section 5 presented code construction and a decoding algorithm to fight against deletions (of runs). Since deletions, insertions, and substitutions can all simultaneously occur in the process [38], one possible open problem is to extend the current framework and error-correcting codes to simultaneously correct three types of errors while still achieving a high writing rate.

Appendix A

Proof of Lemma 78

Lemma 78. *Given $q \geqslant 3$, we have*

$$\max_{\boldsymbol{t} \in \Sigma_q^5} \|R(\mathcal{D}^1(\boldsymbol{t}))\| \leqslant \|R(\mathcal{D}^1(01234))\|,$$

where $\mathcal{D}^1(01234) \subseteq \Sigma_{q+4}^*$ (the substituted symbol can be replaced with another symbol from Σ_{q+4}).

To prove Lemma 78, we start with the definition of dominance between two sequences from [83].

Definition 133. *Let $\boldsymbol{s}$ and $\bar{\boldsymbol{s}}$ be strings of length n, and let A be the set of symbols in $\boldsymbol{s}$ and $\bar{A}$ the set of symbols in $\bar{\boldsymbol{s}}$. We say that $\boldsymbol{s}$ dominates $\bar{\boldsymbol{s}}$ if there exists a function $\eta : A \to \bar{A}$ such that $\bar{\boldsymbol{s}} = \eta(\boldsymbol{s})$, where $\eta(\boldsymbol{s}) = \eta(s_1) \cdots \eta(s_n)$. Furthermore, a set U of strings dominates a set T if there is a single mapping η such that for each string $\boldsymbol{t} \in T$ there is a string $\boldsymbol{u} \in U$ such that $\boldsymbol{t} = \eta(\boldsymbol{u})$.*

For example, 0102 dominates 1212 (using the mapping $\eta(0) = 1, \eta(1) = 2, \eta(2) = 2$) but 0102 does not dominate 0010. The string $012 \cdots k$ dominates any string of length $k + 1$.

We recall an auxiliary lemma showing properties of dominance from [83], along with two other auxiliary lemmas that are used to simplify the proof of Lemma 78.

Lemma 134. *([83, Lemma 1]) Assume there are two strings $\boldsymbol{s}, \bar{\boldsymbol{s}}$ with $\boldsymbol{s}$ dominating $\bar{\boldsymbol{s}}$.*

1. *Suppose we apply the same duplication in both $\boldsymbol{s}$ and $\bar{\boldsymbol{s}}$ (that is, in the same position and with the same length). Let the resulting strings be $\boldsymbol{s}'$ and $\bar{\boldsymbol{s}}'$, respectively. Then $\boldsymbol{s}'$ dominates $\bar{\boldsymbol{s}}'$.*

2. *If a deduplication is possible in $\boldsymbol{s}$, a deduplication in the same position and with the same length is possible in $\bar{\boldsymbol{s}}$. Let the result of applying this deduplication to $\boldsymbol{s}$ and $\bar{\boldsymbol{s}}$ be denoted by $\boldsymbol{s}'$ and $\bar{\boldsymbol{s}}'$, respectively. Then $\boldsymbol{s}'$ dominates $\bar{\boldsymbol{s}}'$.*

Lemma 135. *Let $\bar{\boldsymbol{s}}$ be a string over $\bar{\Sigma}$ and $\boldsymbol{s}$ a string over Σ such that $\boldsymbol{s}$ dominates $\bar{\boldsymbol{s}}$. Let the number of distinct symbols in $\bar{\boldsymbol{s}}$ and $\boldsymbol{s}$ be denoted $\bar{q}_s$ and q_s, respectively, and suppose $\|\Sigma\| \geqslant \|\bar{\Sigma}\| + (q_s - \bar{q}_s)$. Then $\mathcal{D}^p(\boldsymbol{s}) \subseteq \Sigma^*$ dominates $\mathcal{D}^p(\bar{\boldsymbol{s}}) \subseteq \bar{\Sigma}^*$. In other words, there is a mapping $\eta : \Sigma \to \bar{\Sigma}$ that for any $\bar{\boldsymbol{y}} \in \mathcal{D}^p(\bar{\boldsymbol{s}}) \subseteq \bar{\Sigma}^*$, there exists $\boldsymbol{y} \in \mathcal{D}^p(\boldsymbol{s}) \subseteq \Sigma^*$ such that $\bar{\boldsymbol{y}} = \eta(\boldsymbol{y})$.*

Before proving the lemma, we provide an example with multiple short duplications and a substitution error, where the duplicated substrings are marked with underlines and the substituted symbols are in red.

Let $\Sigma = \{0,1,2,3,4\}$ and $\bar{\Sigma} = \{0,1,2,3\}$. Suppose $\boldsymbol{s} = 012$ and $\bar{\boldsymbol{s}} = 010$ with $q_s = 3$ and $\bar{q}_s = 2$. The mapping $\eta(0) = 0$, $\eta(1) = 1$, and $\eta(2) = 0$, shows that $\boldsymbol{s}$ dominates $\bar{\boldsymbol{s}}$, i.e., $\boldsymbol{s} = 012 \to \bar{\boldsymbol{s}} = 010$.

Let $\bar{\boldsymbol{y}}_1 = 010\underline{010010} \in \mathcal{D}(\bar{\boldsymbol{s}})$. Then there exists $\boldsymbol{y}_1 = 012\underline{012012} \in \mathcal{D}(\boldsymbol{s})$ dominating $\bar{\boldsymbol{y}}_1$, via the same mapping η.

Next, assume $\bar{\boldsymbol{y}}_2 = 010012010$ is generated from $\bar{\boldsymbol{y}}_1$ by a substitution $0 \to 2$. Then $\boldsymbol{y}_2 = 012013012$, obtained from $\boldsymbol{y}_1$ after a substitution $2 \to 3$ in the same position, dominates $\bar{\boldsymbol{y}}_2$, via the mapping η extended by $\eta(3) = 2$.

Proof of Lemma 135. Without loss of generality, assume that $\bar{\Sigma} = \{0,1,\ldots,\|\bar{\Sigma}\|-1\}$ and that the symbols appearing in $\bar{\boldsymbol{s}}$ are $0,1,\ldots,\bar{q}_s - 1$, where $\bar{q}_s \leqslant \|\bar{\Sigma}\|$. Similar statements hold for $\Sigma, \boldsymbol{s}, q_s$. By assumption, there exists some mapping $\eta : \{0,\ldots,q_s-1\} \to \{0,\ldots,\bar{q}_s-1\}$ showing that $\boldsymbol{s}$ dominates $\bar{\boldsymbol{s}}$. Since $\|\Sigma\| - q_s \geqslant \|\bar{\Sigma}\| - \bar{q}_s$, we may extend η by mapping symbols in Σ not occurring in $\boldsymbol{s}$ to symbols in $\bar{\Sigma}$ not occurring in $\bar{s}$. Specifically, we assign $\eta(i) = i - (q_s - \bar{q}_s) \in \bar{\Sigma}$ for $i \in \{q_s, q_s+1,\ldots,\|\Sigma\|-1\} \subseteq \Sigma$ to construct $\eta : \Sigma \to \bar{\Sigma}$.

Let the sequence of errors transforming $\bar{\boldsymbol{s}}$ to $\bar{\boldsymbol{y}}$ be denoted by $\bar{T}_j, j = 1,\ldots,k$ and let $\bar{\boldsymbol{y}}_j = \bar{T}_j(\bar{\boldsymbol{y}}_{j-1})$ with $\bar{\boldsymbol{y}}_0 = \bar{\boldsymbol{s}}$ and $\bar{\boldsymbol{y}} = \bar{\boldsymbol{y}}_k$. We will find a corresponding sequence (T_j), where each T_j has the same type of error as $\bar{T}_j$, and define $\boldsymbol{y}_j = T_j(\boldsymbol{y}_{j-1})$. We prove that for each j, we have $\bar{\boldsymbol{y}}_j = \eta(\boldsymbol{y}_j)$. The claim holds for $j = 0$ by assumption. Suppose it holds for $j-1$. We show that it also holds for j. If $\bar{T}_j$ is a duplication, by Lemma 134.1), then we choose T_j to be a duplication of the same length in the same position. If $\bar{T}_j$ substitutes some symbol in $\bar{\boldsymbol{y}}_{j-1}$ with $a \in \bar{\Sigma}$, then T_j substitutes the symbol in the same position in $\boldsymbol{y}_{j-1}$ with a symbol $b \in \Sigma$ such that $\eta(b) = a$. It then follows that $\bar{\boldsymbol{y}}_j = \eta(\boldsymbol{y}_j)$ for each $\bar{\boldsymbol{y}}_j$. Therefore, we have $\mathcal{D}^p(\boldsymbol{s}) \subseteq \Sigma^*$ dominates $\mathcal{D}^p(\bar{\boldsymbol{s}}) \subseteq \bar{\Sigma}^*$. □

Lemma 136. *If a set of strings Y dominates a second set $\bar{Y}$, then $||R(\bar{Y})|| \leqslant ||R(Y)||$.*

Proof. Suppose Y dominates $\bar{Y}$ via a mapping $\eta : \Sigma \to \bar{\Sigma}$. Then, for each $\bar{\boldsymbol{y}} \in \bar{Y}$, there exists some $\boldsymbol{y} \in Y$ such that $\bar{\boldsymbol{y}} = \eta(\boldsymbol{y})$. For $\bar{\boldsymbol{y}} \in \bar{Y}$, define $\eta^{-1}(\bar{\boldsymbol{y}})$ as the lexicographically-smallest sequence among $\{\boldsymbol{y} \in Y : \eta(\boldsymbol{y}) = \bar{\boldsymbol{y}}\}$. Furthermore, define $Y' = \{\eta^{-1}(\bar{\boldsymbol{y}}) : \bar{\boldsymbol{y}} \in \bar{Y}\}$ and note that $Y' \subseteq Y$. With this definition, Y' dominates $\bar{Y}$ and η is a bijection between the two sets. We have $\|\bar{Y}\| = \|Y'\| \leqslant \|Y\|$. Also, as $Y' \subseteq Y$, we have $\|R(Y')\| \leqslant \|R(Y)\|$.

To prove the lemma, we show that $\|R(\bar{Y})\| \leqslant \|R(Y')\|$. It suffices to prove that if $\bar{\boldsymbol{y}}_1, \bar{\boldsymbol{y}}_2 \in \bar{Y}$ have distinct roots, then $\boldsymbol{y}_1, \boldsymbol{y}_2 \in Y'$, where $\boldsymbol{y}_1 = \eta^{-1}(\bar{\boldsymbol{y}}_1)$ and $\boldsymbol{y}_2 = \eta^{-1}(\bar{\boldsymbol{y}}_2)$, also have distinct roots.

Suppose, on the contrary, that $\boldsymbol{y}_1, \boldsymbol{y}_2$ do not have distinct roots, i.e., $R(\boldsymbol{y}_1) = R(\boldsymbol{y}_2)$. Let T_1 and T_2 represent the sequences of deduplications on $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ that produce their roots, i.e., $R(\boldsymbol{y}_1) = T_1(\boldsymbol{y}_1)$ and $R(\boldsymbol{y}_2) = T_2(\boldsymbol{y}_2)$. Based on the Lemma 134.2) above, there exist two corresponding sequences of deduplications $\bar{T}_1$ and $\bar{T}_2$ such that $\bar{T}_1(\bar{\boldsymbol{y}}_1) = \eta(R(\boldsymbol{y}_1))$ and $\bar{T}_2(\bar{\boldsymbol{y}}_2) = \eta(R(\boldsymbol{y}_2))$. If $R(\boldsymbol{y}_1) = R(\boldsymbol{y}_2)$, then $\bar{T}_1(\bar{\boldsymbol{y}}_1) = \bar{T}_2(\bar{\boldsymbol{y}}_2)$. But by the uniqueness of the root, $R(\bar{\boldsymbol{y}}_1) = R(\bar{T}_1(\bar{\boldsymbol{y}}_1))$ and

$R(\bar{\boldsymbol{y}}_2) = R(\bar{T}_2(\bar{\boldsymbol{y}}_2))$. So $R(\bar{\boldsymbol{y}}_1) = R(\bar{\boldsymbol{y}}_2)$. But this contradicts the assumption. Hence, the roots of $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ are distinct. □

With Lemma 135 and Lemma 136 in hand, we prove Lemma 78 in the following.

Proof of Lemma 78. Let $\boldsymbol{s} = 01234$. If $\boldsymbol{t}$ is the empty string, the claim is trivial. So in the rest of the proof, we assume $\boldsymbol{t}$ is not empty. Based on Definition 133, $\boldsymbol{s}$ dominates $\boldsymbol{t}$ for any $\boldsymbol{t} \in \Sigma_q^5 \setminus \{\Lambda\}$. Let q_t denote the number of distinct symbols in $\boldsymbol{t}$ and note that there are 5 distinct symbols in $\boldsymbol{s}$. By Lemma 135, with $p = 1$, $\mathcal{D}^1(\boldsymbol{s}) \subseteq \Sigma_{q+4}^*$ dominates $\mathcal{D}^1(\boldsymbol{t}) \subseteq \Sigma_q^*$ for any $\boldsymbol{t} \in \Sigma_q^5$ since $q+4 \geqslant q+(5-q_t)$ as $q_t \geqslant 1$. Applying Lemma 136 to $\mathcal{D}^1(\boldsymbol{s})$ and $\mathcal{D}^1(\boldsymbol{t})$ completes the proof. □

Bibliography

[1] M. Abroshan, R. Venkataramanan, L. Dolecek, and A. G. i Fabregas, "Coding for deletion channels with multiple traces", in *2019 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2019, pp. 1372–1376.

[2] R. Bitar, S. K. Hanna, N. Polyanskii, and I. Vorobyev, "Optimal codes correcting localized deletions", in *2021 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2021, pp. 1991–1996.

[3] R. E. Blahut, *Algebraic codes for data transmission*. Cambridge university press, 2003.

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

[5] J. Brakensiek, R. Li, and B. Spang, "Coded trace reconstruction in a constant number of traces", in *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, IEEE, 2020, pp. 482–493.

[6] K. Cai, Y. M. Chee, R. Gabrys, H. M. Kiah, and T. T. Nguyen, "Optimal codes correcting a single indel/edit for DNA-based data storage", *arXiv preprint arXiv:1910.06501*, 2019.

[7] Y. M. Chee, J. Chrisnata, H. M. Kiah, and T. T. Nguyen, "Deciding the confusability of words under tandem repeats in linear time", *ACM Transactions on Algorithms (TALG)*, vol. 15, no. 3, pp. 1–22, 2019.

[8] Y. M. Chee, J. Chrisnata, H. M. Kiah, and T. T. Nguyen, "Efficient encoding/decoding of GC-balanced codes correcting tandem duplications", *IEEE Transactions on Information Theory*, vol. 66, no. 8, pp. 4892–4903, 2020.

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

[10] M. Cheraghchi, R. Gabrys, O. Milenkovic, and J. Ribeiro, "Coded trace reconstruction", *IEEE Transactions on Information Theory*, vol. 66, no. 10, pp. 6084–6103, 2020.

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

[12] D. Deamer, M. Akeson, and D. Branton, "Three decades of nanopore sequencing", *Nature Biotechnology*, vol. 34, no. 5, pp. 518–524, May 2016.

[13] A. Doricchi, C. M. Platnich, A. Gimpel, *et al.*, "Emerging approaches to dna data storage: Challenges and prospects", *ACS nano*, vol. 16, no. 11, pp. 17 552–17 571, 2022.

[14] O. Elishco, R. Gabrys, and E. Yaakobi, "Bounds and constructions of codes over symbol-pair read channels", *IEEE Transactions on Information Theory*, vol. 66, no. 3, pp. 1385–1395, 2020.

[15] F. Farnoud, M. Schwartz, and J. Bruck, "Estimation of duplication history under a stochastic model for tandem repeats", *BMC Bioinformatics*, vol. 20, no. 1, 2019.

[16] F. Farnoud, M. Schwartz, and J. Bruck, "The capacity of string-duplication systems", *IEEE Transactions on Information Theory*, vol. 62, no. 2, pp. 811–824, 2015.

[17] P. Fire, *A class of multiple-error-correcting binary codes for non-independent errors*. Stanford University, 1959, vol. 55.

[18] R. Gabrys, S. Pattabiraman, and O. Milenkovic, "Mass error-correction codes for polymer-based data storage", in *IEEE International Symposium on Information Theory (ISIT)*, 2020, pp. 25–30.

[19] R. Gabrys, E. Yaakobi, and O. Milenkovic, "Codes in the Damerau distance for deletion and adjacent transposition correction", *IEEE Transactions on Information Theory*, vol. 64, no. 4, pp. 2550–2570, 2017.

[20] P. Gaytán, "Chemical synthesis of oligonucleotides using acetone as a washing solvent", *Biotechniques*, vol. 47, no. 2, pp. 701–702, 2009.

[21] M. George, S. Jafarpour, and F. Bullo, "Markov chains with maximum entropy for robotic surveillance", *IEEE Transactions on Automatic Control*, vol. 64, no. 4, pp. 1566–1580, 2018.

[22] N. Goldman, P. Bertone, S. Chen, *et al.*, "Towards practical, high-capacity, low-maintenance information storage in synthesized DNA", *Nature*, vol. 494, no. 7435, pp. 77–80, 2013.

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

[24] S. K. Hanna and S. El Rouayheb, "Codes for correcting localized deletions", *IEEE Transactions on Information Theory*, vol. 67, no. 4, pp. 2206–2216, 2021.

[25] A. P. Heikema, D. Horst-Kreft, S. A. Boers, *et al.*, "Comparison of illumina versus nanopore 16s rrna gene sequencing of the human nasal microbiota", *Genes*, vol. 11, no. 9, p. 1105, 2020.

[26] T. Holenstein, M. Mitzenmacher, R. Panigrahy, and U. Wieder, "Trace reconstruction with constant deletion probability and related results.", in *SODA*, vol. 8, 2008, pp. 389–398.

[27] S. e. IDC, "Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2020, with forecasts from 2021 to 2025", *IDC Statista*, 2021.

[28] S. Jain, F. Farnoud, M. Schwartz, and J. Bruck, "Coding for optimized writing rate in DNA storage", in *IEEE International Symposium on Information Theory (ISIT)*, 2020, pp. 711–716.

[29] S. Jain, F. Farnoud, and J. Bruck, "Capacity and expressiveness of genomic tandem duplication", *IEEE Transactions on Information Theory*, vol. 63, no. 10, pp. 6129–6138, 2017.

[30] S. Jain, F. Farnoud, M. Schwartz, and J. Bruck, "Duplication-correcting codes for data storage in the DNA of living organisms", *IEEE Transactions on Information Theory*, vol. 63, no. 8, pp. 4996–5010, 2017.

[31] S. Jain, F. Farnoud, M. Schwartz, and J. Bruck, "Noise and uncertainty in string-duplication systems", in *2017 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2017, pp. 3120–3124.

[32] S. Jain, F. F. Hassanzadeh, M. Schwartz, and J. Bruck, "Duplication-correcting codes for data storage in the dna of living organisms", *IEEE Transactions on Information Theory*, vol. 63, no. 8, pp. 4996–5010, 2017.

[33] M. A. Jensen and R. W. Davis, "Template-independent enzymatic oligonucleotide synthesis (tieos): Its history, prospects, and challenges", *Biochemistry*, vol. 57, no. 12, pp. 1821–1832, 2018.

[34] H. M. Kiah, T. Thanh Nguyen, and E. Yaakobi, "Coding for sequence reconstruction for single edits", in *IEEE International Symposium on Information Theory (ISIT)*, 2020, pp. 676–681.

[35] M. Kovačević, "Codes correcting all patterns of tandem-duplication errors of maximum length 3", *arXiv preprint arXiv:1911.06561*, 2019.

[36] M. Kovačević and V. Y. Tan, "Asymptotically optimal codes correcting fixed-length duplication errors in DNA storage systems", *IEEE Communications Letters*, vol. 22, no. 11, pp. 2194–2197, 2018.

[37] E. S. Lander, L. M. Linton, B. Birren, *et al.*, "Initial sequencing and analysis of the human genome", 2001.

[38] H. H. Lee, R. Kalhor, N. Goela, J. Bolot, and G. M. Church, "Terminator-free template-independent enzymatic DNA synthesis for digital information storage", *Nature communications*, vol. 10, no. 1, pp. 1–12, 2019.

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

[40] A. Lenz, Y. Liu, C. Rashtchian, P. H. Siegel, A. Wachter-Zeh, and E. Yaakobi, "Coding for efficient DNA synthesis", in *IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2020, pp. 2885–2890.

[41] A. Lenz, I. Maarouf, L. Welter, A. Wachter-Zeh, E. Rosnes, and A. G. i Amat, "Concatenated codes for recovery from multiple reads of DNA sequences", in *2020 IEEE Information Theory Workshop (ITW)*, IEEE, 2021, pp. 1–5.

[42] A. Lenz and N. Polyanskii, "Optimal codes correcting a burst of deletions of variable length", in *2020 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2020, pp. 757–762.

[43] A. Lenz, A. Wachter-Zeh, and E. Yaakobi, "Duplication-Correcting Codes", *arXiv:1712.09345 [cs, math]*, Dec. 2017. arXiv: `1712.09345 [cs, math]`.

[44] A. Lenz, A. Wachter-Zeh, and E. Yaakobi, "Duplication-correcting codes", *Designs, Codes and Cryptography*, vol. 87, pp. 277–298, 2019.

[45] E. M. LeProust, B. J. Peck, K. Spirin, *et al.*, "Synthesis of high-quality libraries of long (150mer) oligonucleotides by a novel depurination controlled process", *Nucleic acids research*, vol. 38, no. 8, pp. 2522–2540, 2010.

[46] P. Leupold, V. Mitrana, and J. M. Sempere, "Formal languages arising from gene repeated duplication", in *Aspects of Molecular Computing*, Springer, 2003, pp. 297–308.

[47] Y. Li, S. Wang, C. Bi, Z. Qiu, M. Li, and X. Gao, "Deepsimulator1. 5: A more powerful, quicker and lighter simulator for nanopore sequencing", *Bioinformatics*, vol. 36, no. 8, pp. 2578–2580, 2020.

[48] D. Lind, B. Marcus, L. Douglas, and M. Brian, *An introduction to symbolic dynamics and coding*. Cambridge university press, 1995.

[49] H. Mahdavifar and A. Vardy, "Asymptotically optimal sticky-insertion-correcting codes with efficient encoding and decoding", in *2017 IEEE International Symposium on Information Theory (ISIT)*, Jun. 2017, pp. 2683–2687.

[50] B. H. Marcus, R. M. Roth, and P. H. Siegel, "An introduction to coding for constrained systems", *Lecture notes*, 2001.

[51] C. R. Mehta and N. R. Patel, "IBM SPSS exact tests", *Armonk, NY: IBM Corporation*, pp. 23–24, 2011.

[52] L. C. Meiser, P. L. Antkowiak, J. Koch, *et al.*, "Reading and writing digital data in DNA", *Nature Protocols*, vol. 15, no. 1, pp. 86–101, 2020.

[53] O. Milenkovic and C. Pan, "Dna-based data storage systems: A review of implementations and code constructions", *arXiv preprint arXiv:2310.04694*, 2023.

[54] F. Nazarov and Y. Peres, "Trace reconstruction with $\exp(o(n1/3))$ samples", in *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, 2017, pp. 1042–1046.

[55] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins", *Journal of molecular biology*, vol. 48, no. 3, pp. 443–453, 1970.

[56] T. T. Nguyen, K. Cai, K. A. S. Immink, and H. M. Kiah, "Capacity-approaching constrained codes with error correction for dna-based data storage", *IEEE Transactions on Information Theory*, vol. 67, no. 8, pp. 5602–5613, 2021.

[57] T. T. Nguyen, K. Cai, K. A. S. Immink, and H. M. Kiah, "Constrained coding with error control for DNA-based data storage", in *IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2020, pp. 694–699.

[58] T. T. Nguyen, K. Cai, and P. H. Siegel, "Every bit counts: A new version of non-binary vt codes with more efficient encoder", in *ICC 2023-IEEE International Conference on Communications*, IEEE, 2023, pp. 5477–5482.

[59] L. Organick, S. D. Ang, Y.-J. Chen, *et al.*, "Random access in large-scale DNA data storage", *Nature biotechnology*, vol. 36, no. 3, pp. 242–248, 2018.

[60] A. Orlitsky, "Interactive communication: Balanced distributions, correlated files, and average-case complexity", in *[1991] Proceedings 32nd Annual Symposium of Foundations of Computer Science*, Oct. 1991, pp. 228–238.

[61] S.-J. Park, H. Park, H.-Y. Kwak, and J.-S. No, "Bic codes: Bit insertion-based constrained codes with error correction for dna storage", *IEEE Transactions on Emerging Topics in Computing*, 2023.

[62] D. Pumpernik, B. Oblak, and B. Borštnik, "Replication slippage versus point mutation rates in short tandem repeats of the human genome", *Molecular Genetics and Genomics*, vol. 279, no. 1, pp. 53–61, 2008.

[63] R. M. Roth, "Introduction to coding theory", *IET Communications*, vol. 47, 2006.

[64] F. Sala, R. Gabrys, C. Schoeny, and L. Dolecek, "Exact reconstruction from insertions in synchronization codes", *IEEE Transactions on Information Theory*, vol. 63, no. 4, pp. 2428–2445, 2017.

[65] C. Schoeny, F. Sala, and L. Dolecek, "Novel combinatorial coding results for DNA sequencing and data storage", in *2017 51st Asilomar Conference on Signals, Systems, and Computers*, IEEE, 2017, pp. 511–515.

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

[67] M. Schwartz and J. Bruck, "On the capacity of the precision-resolution system", *IEEE transactions on information theory*, vol. 56, no. 3, pp. 1028–1037, 2010.

[68] R. Shafir, O. Sabary, L. Anavy, E. Yaakobi, and Z. Yakhini, "Sequence reconstruction under stutter noise in enzymatic dna synthesis", in *2021 IEEE Information Theory Workshop (ITW)*, IEEE, 2021, pp. 1–6.

[69] S. L. Shipman, J. Nivala, J. D. Macklis, and G. M. Church, "CRISPR–Cas encoding of a digital movie into the genomes of a population of living bacteria", en, *Nature*, vol. 547, no. 7663, pp. 345–349, Jul. 2017.

[70] S. L. Shipman, J. Nivala, J. D. Macklis, and G. M. Church, "Molecular recordings by directed CRISPR spacer acquisition", en, *Science*, Jun. 2016.

[71] J. Sima, "Correcting errors in dna storage", Ph.D. dissertation, California Institute of Technology, 2022.

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

[73] J. Sima, R. Gabrys, and J. Bruck, "Optimal codes for the q-ary deletion channel", in *2020 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2020, pp. 740–745.

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

[75] J. Sima, R. Gabrys, and J. Bruck, "Syndrome compression for optimal redundancy codes", in *2020 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2020, pp. 751–756.

[76] J. Sima, N. Raviv, and J. Bruck, "Robust indexing-optimal codes for DNA storage", in *IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2020, pp. 717–722.

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

[78] S. R. Srinivasavaradhan, S. Gopi, H. D. Pfister, and S. Yekhanin, "Trellis BMA: Coded trace reconstruction on IDS channels for DNA storage", in *2021 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2021, pp. 2453–2458.

[79] Y. Tang, Y. Yehezkeally, M. Schwartz, and F. Farnoud, "Single-error detection and correction for duplication and substitution channels", *IEEE Transactions on Information Theory*, vol. 66, no. 11, pp. 6908–6919, 2020.

[80] Y. Tang and F. Farnoud, "Correcting deletion errors in DNA data storage with enzymatic synthesis", in *2021 IEEE Information Theory Workshop (ITW)*, 2021, pp. 1–6.

[81] Y. Tang and F. Farnoud, "Error-correcting codes for noisy duplication channels", *IEEE Transactions on Information Theory*, vol. 67, no. 6, pp. 3452–3463, 2021.

[82] Y. Tang and F. Farnoud, "Error-correcting codes for noisy duplication channels", in *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, IEEE, 2019, pp. 140–146.

[83] Y. Tang and F. Farnoud, "Error-correcting codes for short tandem duplication and edit errors", *IEEE Transactions on Information Theory*, vol. 68, no. 2, pp. 871–880, 2022.

[84] Y. Tang and F. Farnoud, "Error-correcting codes for short tandem duplication and substitution errors", in *IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2020, pp. 734–739.

[85] Y. Tang and F. Farnoud, "Error-correcting codes for short tandem duplication and substitution errors", *arXiv preprint arXiv:2011.05896*, 2020.

[86] Y. Tang, H. Lou, and F. Farnoud, "Error-correcting codes for short tandem duplications and at most p substitutions", in *2021 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2021, pp. 1835–1840.

[87] Y. Tang, S. Motamen, H. Lou, *et al.*, "Correcting a substring edit error of bounded length", in *2023 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2023, pp. 2720–2725.

[88] Y. Tang, S. Wang, R. Gabrys, and F. Farnoud, "Correcting multiple short-duplication and substitution errors", in *2022 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2022, pp. 1–6.

[89] Y. Tang, S. Wang, H. Lou, R. Gabrys, and F. Farnoud, "Low-redundancy codes for correcting multiple short-duplication and edit errors", *IEEE Transactions on Information Theory*, vol. 69, no. 5, pp. 2940–2954, 2023.

[90] Y. Tang, Y. Yehezkeally, M. Schwartz, and F. F. Hassanzadeh, "Single-error detection and correction for duplication and substitution channels", in *2019 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2019, pp. 300–304.

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

[92] L. M. Tolhuizen, "The generalized gilbert-varshamov bound is implied by turan's theorem [code construction]", *IEEE Transactions on Information Theory*, vol. 43, no. 5, pp. 1605–1606, 1997.

[93] S. Wang, Y. Tang, R. Gabrys, and F. Farnoud, "Permutation codes for correcting a burst of at most t deletions", in *2022 58th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, IEEE, 2022, pp. 1–6.

[94] S. Wang, Y. Tang, J. Sima, R. Gabrys, and F. Farnoud, *Non-binary codes for correcting a burst of at most t deletions*, 2022.

[95] S. Wang, Y. Tang, J. Sima, R. Gabrys, and F. Farnoud, "Non-binary codes for correcting a burst of at most t deletions", *IEEE Transactions on Information Theory*, 2023.

[96] Z. Yan, C. Liang, and H. Wu, "Upper and lower bounds on the capacity of the dna-based storage channel", *IEEE Communications Letters*, vol. 26, no. 11, pp. 2586–2590, 2022.

[97] S. H. T. Yazdi, R. Gabrys, and O. Milenkovic, "Portable and error-free DNA-based data storage", *Scientific reports*, vol. 7, no. 1, pp. 1–6, 2017.

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

[99] S. H. T. Yazdi, Y. Yuan, J. Ma, H. Zhao, and O. Milenkovic, "A rewritable, random-access DNA-based storage system", *Scientific reports*, vol. 5, no. 1, pp. 1–10, 2015.

[100] Y. Yehezkeally and M. Schwartz, "Uncertainty of reconstructing multiple messages from uniform-tandem-duplication noise", in *IEEE International Symposium on Information Theory (ISIT)*, 2020, pp. 126–131.

[101] Y. Yehezkeally and M. Schwartz, "Reconstruction codes for DNA sequences with uniform tandem-duplication errors", *IEEE Transactions on Information Theory*, vol. 66, no. 5, pp. 2658–2668, 2020.

[102] S. S. Yim, R. M. McBee, A. M. Song, Y. Huang, R. U. Sheth, and H. H. Wang, "Robust direct digital-to-biological data storage in living cells", *Nature chemical biology*, vol. 17, no. 3, pp. 246–253, 2021.

[103] K. Zhou, A. Aertsen, and C. W. Michiels, "The role of variable dna tandem repeats in bacterial adaptation", *FEMS microbiology reviews*, vol. 38, no. 1, pp. 119–141, 2014.

[104] W. Zhou, S. Lin, and K. Abdel-Ghaffar, "Burst or random error correction based on Fire and BCH codes", in *2014 Information Theory and Applications Workshop (ITA)*, IEEE, 2014, pp. 1–5.