Codes correcting bursts of deletions/insertions and tandem duplications

Nguyen, Tuan Thanh

2018

Nguyen, T. T. (2018). Codes correcting bursts of deletions/insertions and tandem duplications. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/89953>

<https://doi.org/10.32657/10220/47179>

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

**CODES CORRECTING BURSTS OF
DELETIONS/INSERTIONS AND TANDEM
DUPLICATIONS**

NGUYEN TUAN THANH

School of Physical and Mathematical Sciences

2018

**CODES CORRECTING BURSTS OF
DELETIONS/INSERTIONS AND TANDEM
DUPLICATIONS**

NGUYEN TUAN THANH

School of Physical and Mathematical Sciences

**A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Doctor of Philosophy**

2018

Abstract

Error-correcting codes have played an important role in improving efficiency and reliability in classical communication and data storage systems. This thesis is devoted to the design of error-correcting codes to combat the most crucial errors that arose from modern communication and data storage systems such as *deletions*, *insertions*, *duplications* (a special type of insertions), and we refer them as *codes correcting edits*.

Designing codes correcting deletion and insertion errors has been the subject of a large body of works in the literature and to the best of our knowledge, there exist only constructions for optimal codes capable of correcting a single deletion or single insertion in permutation codes and q -ary codes. The research on codes combatting multiple errors is very limited. The main objective of this thesis is to design codes capable of correcting *burst errors*, i.e. those errors that occur at adjacent positions. Throughout this thesis, we propose several constructions of burst deletion/insertion-correcting codes over permutations, multipermutations or more generally, over a q -ary alphabet. Special attention is given to the case of burst deletion-correcting codes in permutations and multipermutations. Here, such codes improve the reliability and memory endurance of non-volatile memories such as flash memories. Besides code constructions, we also provide efficient error decoders to recover codewords from errors with linear-time complexity.

In addition, we introduce codes correcting *tandem duplications*, a special type of insertion errors, that has applications that store data in living organisms. We are

concerned with designing efficient channel encoders (or source decoders) that encode (decode) any arbitrary sequence (arbitrary codeword) to a corresponding codeword in our design codes (to the original sequence) with the purpose of achieving high rate and low complexity. Our encoding/decoding algorithms are based on enumerative coding, finite-state machine, and algebraic number theory. While these techniques are standard in constrained coding and combinatorics literature, our contribution is a detailed analysis of the space and time complexities of the respective algorithms.

Acknowledgements

To pursue and accomplish a doctorate degree has been my goal since I was in high school. This has been a long journey and it would not be possible without the help of many people. I am grateful to all of those that I have had the pleasure to work with throughout my study in Nanyang Technological University, Singapore. I would like to take this opportunity to say thank to them.

I would like to express my sincere gratitude to my main supervisor, Prof. Yeow Meng Chee for his guidance and support during the past four years. It is my pleasure and honor to accomplish my PhD studies under his supervision. Yeow Meng has taught me to think deeply about problems and always reminded me that *quality is better than quantity* when doing research. He gave me many interesting problems, and encouraged me to fight hard to achieve the goal. I also would like to thank him for his financial support that allowed me to attend many interesting conferences and workshops.

I would like to thank my co-supervior, Prof. Han Mao Kiah for his helpful suggestions, and especially his patient guidance in writing. He helped me a lot with the revisions of our papers. I am indebted to him for his kindness, understanding in both academic and personal aspects.

My PhD experience has been wonderful with many collaborators. I would like to thank them for their helpful discussions and cooperations: Prof. San Ling, Dr. Xiande Zhang, Dr. Hengjia Wei, Dr. Van Khu Vu, and Johan Chrisnata. I also

want to thank all the members of my research lab SPMS-MAS-04-06 for sharing their knowledge and experience. A special thanks to Dr. Frederic Ezerman, who is always so willing to help and support.

It is my pleasure to be a teaching assistant with Prof. Xiaohui Bei, Dr. Wee Kee Tang, and Prof. Han Mao Kiah. I have learned a lot from their teaching experiences to make every lesson interactive, to get students involved by designing interesting activities in class.

I am extremely grateful to my parents for their great love, encouragement, support, and all invaluable contributions to my life.

Last but not least, words simply cannot express my love and gratitude to my wife, Bao Yen Nguyen, who has always been beside me through hard time. When I started my final year of Phd, she chose to leave her dream job as a flight attendant in Taiwan to come to Singapore so that she can take care of me and cook for me everyday. Without her, this thesis would certainly take longer time to complete. A very special thank to my mother in law for taking care of my wife when she is pregnant during the writing of this thesis.

To my parents and my wife,
To Bun,
I love you!

List of Works

Below is the list of works done during my PhD studies in Nanyang Technological University, in chronological order.

1. Yeow Meng Chee, Johan Chrisnata, Han Mao Kiah, San Ling, **Tuan Thanh Nguyen**, and Van Khu Vu, “Efficient encoding/decoding of capacity-achieving constant-composition ICI-free codes”, in *Proceedings of the 2016 IEEE International Symposium on Information Theory*, pp. 205-209, 2016.
2. Yeow Meng Chee, Han Mao Kiah, San Ling, **Tuan Thanh Nguyen**, Van Khu Vu, and Xiande Zhang, “String concatenation construction for Chebyshev permutation channel codes”, in *Proceedings of the 2016 IEEE International Symposium on Information Theory*, pp. 2824-2828, 2016.
3. Yeow Meng Chee, San Ling, **Tuan Thanh Nguyen**, Van Khu Vu, and Hengjia Wei, “Permutation codes correcting a single burst deletion II: stable deletions”, in *Proceedings of the 2017 IEEE International Symposium on Information Theory*, pp. 2693-2697, 2017.
4. Yeow Meng Chee, Johan Chrisnata, Han Mao Kiah, and **Tuan Thanh Nguyen**, “Efficient encoding/decoding of irreducible words for codes correcting tandem duplications”, in *Proceedings of the 2018 IEEE International Symposium on Information Theory*, to appear (accepted March 2018).
5. Yeow Meng Chee, Johan Chrisnata, Han Mao Kiah, **Tuan Thanh Nguyen**,

“Codes for Correcting Tandem Repeats”, in *9th Annual Non-Volatile Memories Workshop*, 2018.

6. Yeow Meng Chee, Johan Chrisnata, Han Mao Kiah, **Tuan Thanh Nguyen**,
“Deciding the Confusability of Words under Tandem Repeats in Linear Time”,
submitted to *ACM Transactions on Algorithms*, April 2018.

Contents

1	Introduction	5
1.1	Deletion/Insertion-Correcting Codes	5
1.2	BDICC for Communications	7
1.3	BDICC for Flash Memories	8
1.4	Duplication-Correcting Codes for DNA Storage	10
1.5	Organization	11
1.6	Bibliographical Notes	12
2	Preliminaries	13
2.1	Basic Notation	13
2.2	Coding Theory	14
2.3	Analysis	15
2.4	Number Theory	15
3	Permutation Codes	16
3.1	Classification of Deletions/Insertions in Permutations	16
3.2	Previous Results	20
3.3	Permutation Codes Correcting a Burst of Stable Deletions	21
3.4	Permutation Codes Correcting a Burst of Stable Insertions	31

3.5	Permutation Codes Correcting a Burst of Unstable Deletions	36
4	Multipermutation and q-ary Codes	39
4.1	Definitions and Notations	39
4.2	s - BD and s - BI multipermutation codes	42
4.3	Multipermutation Codes Correcting a Single Deletion/Insertion	43
4.4	Multipermutation Codes Correcting a Burst of Deletions/Insertions . .	45
4.5	Constructions for General q -ary Codes	48
5	Codes Correcting Tandem Duplications	52
5.1	Definitions and Notations	53
5.2	Unbounded Tandem Duplication Codes	54
5.3	Previous Results and Open Problems	55
5.4	Irreducible Words and Our Conjecture	59
5.5	Efficient Encoding/Decoding for $(n, \leq k; q)$ -TD codes, where $k \in \{2, 3\}$.	65
5.6	The First Known Solution when $k > 3$	75
6	Conclusion and Future Direction	81

Chapter 1

Introduction

1.1 Deletion/Insertion-Correcting Codes

Throughout this thesis, we design codes correcting deletions and insertions with efficient error decoders. While designing codes correcting these errors has been the subject of a large body of works in the literature, the results on codes combatting multiple errors is very limited. In fact, the size of an optimal code that can correct a single deletion or insertion is still unknown.

Deletion/insertion-correcting codes were first introduced by in the 1960s by Varshamov, Tenengolts, and Levenshtein [29, 30]. In 1965, Varshamov and Tenengolts created a special class of binary codes known as **VT-codes** that is capable of correcting asymmetric errors on the Z-channel [32], and later, Levenshtein used these codes to correct a single deletion or insertion and proved that it is asymptotically optimal. In 1984, Tenengolts used VT-codes to construct asymptotically optimal code correcting single deletion or insertion over q -ary alphabet [30].

Deletion/insertion-correcting codes have been investigated in both adversarial channels and probabilistic channels. In the adversarial case, Sloane [10] provided an explicit and simple formula to compute exactly the size of all classes of VT-codes. Abdel-Ghaffar et al. [6] presented an efficient encoding/decoding algorithm for VT-

codes with $\log(n)$ redundancy where n is the length of each binary sequence. More recently, Guruswami et al. [8] construct a binary code to combat k deletions with $c_k \log(n)$ redundancy. They showed that $c_k = O(k^2 \log(k))$ and the codes can be decoded efficiently in $O_k(n \log^4(n))$ time (the definition of big O relation is introduced in Chapter 2). Guruswami et al. [9] also investigate probabilistic deletion channel where each transmitted bit is deleted independently with probability p , for some $p \in (0, 1)$, which they called the deletion probability. They then provided an explicit construction with polynomial time encoding and an error decoder with rate $\epsilon(1 - p)$ for a constant $\epsilon > 0$.

Recently, *burst deletion/insertion-correcting codes* have attracted attention due to various applications in modern communications and storage systems. For example, in wireless communications, burst errors due to multi-path fading occur with high frequency. Binary codes capable of correcting burst of deletions were introduced by Cheng et al. [33] and Schoeny et al. [31]. In addition, recent applications in rank modulation in flash memories have suggested the need to study permutation and multipermutation codes correcting deletions or insertions [19, 22, 24, 26]. However, as in binary case, there exist only constructions for asymptotically optimal codes capable of correcting a single deletion or single insertion in permutation codes and q -ary codes. The research on codes combatting burst errors is very limited.

This thesis is devoted to the design of burst deletion/insertion-correcting codes (BDICC) in permutations, multipermutations, or more generally, over a q -ary alphabet of size more than two. Based on the design of communication and storage systems (as illustrated by Figure 1.1), there are three main criteria in the design of BDICC.

Code size. First, we want to maximize the amount of information that we can communicate through the noisy channel. Therefore, it is important to design a code that contains as many codewords as possible.

Efficient error decoder. In addition, we want an efficient method of correcting

errors whenever they occur.

Efficient Channel Encoder and Source Decoder. For channel encoding, we want efficient methods of mapping a message $\mathbf{m}$ to a codeword $\mathbf{c}$ in $\mathcal{C}$. Conversely, for source decoding, we want efficient methods of mapping $\mathbf{c}$ back to $\mathbf{m}$. For easier exposition, we refer to source decoding as decoding. We desire encoding methods that run in time linear in the length of the message. If this is not known, we also look at algorithms that run in polynomial time.

Figure 1.1: The use of error-correcting codes in communication

For most of the results, error decoding is shown to be efficient. It is not clear on how to efficiently encode a source message. Nevertheless, we provide efficient channel encoders and source decoders for codes correcting tandem duplications in Section 5.

1.2 BDICC for Communications

In many digital communications systems, deletions and insertions are typical errors due to synchronization disturbances such as timing defects or packet-loss. In wireless communications, burst errors due to multi-path fading occur with high frequency.

Our contribution. We study q -ary codes that can correct a burst of deletions or insertions. We provide several constructions and prove that the error-decoders run in time linear in the length of the message. In addition, we also construct q -ary constant-composition burst-deletion/insertion correcting codes. The results are presented in Chapter 4.

1.3 BDICC for Flash Memories

In flash memories, information is stored in arrays of memory cells, and each memory cell may store one or more bits of information. The level of a cell can be increased by injecting charge into the cell, and decreased by removing charge from the cell. While adding charge to a single cell can be done easily, removing charge from a single cell is difficult. It follows from the fact that most flash memory technologies do not allow a single cell to be erased. In order to decrease any cell level, the whole block of memory cells needs to be erased, in other words, we need to remove the charge from all other cells and then reprogrammed the whole block. The two main challenges in flash memories are as follows.

(i) Programing cells: it is not easy to program each cell exactly to its designated level. In fact, the process of writing a specific level on a cell is designed to cautiously approach the target level from below so as to avoid undesired block erasures in case of overshooting. To overcome this problem, in 2009, Jiang et al. [22] proposed a permutation coded rank modulation scheme. In this setup, the information is stored in the form of ranking of the cells' charges rather than in term of the absolute values of the charges. Specifically, information is written in blocks of n cells, and each block represents a permutation of length n (Figure 1.2). This simple coding framework may eliminate the problem of cell block erasures as well as potential cell overshooting issues [23]. This is because even though charge leakage may alter the absolute cell levels, the relative ordering of cells levels remain unchanged. Recently, coded rank modulation scheme has been used to combat other kinds of errors (refer to [23, 25]).

(ii) Memory endurance in aging device: as mentioned above, it is costly to decrease the cell's level as we need to erase and reprogram the whole block. Therefore, it is highly desirable to minimize the number of block erasures. En Gad et al. [26] proposed the use of multipermutations and demonstrated that the flexibility could result in better performance. As before, their proposal uses relative charge levels in

Figure 1.2: Charge levels in a block of nine cells in a flash memory. The information stored in the block is represented by a permutation $(5, 3, 2, 8, 4, 6, 7, 1, 9)$, based on the relative values of the charge levels.

stead of absolute levels, however, their proposal allows multiple cells to be in the same relative level (Figure 1.3).

Figure 1.3: Charge levels in a block of nine cells in a flash memory. The information stored in the block is represented by a multipermutation $(3, 3, 2, 1, 2, 1, 2, 1, 3)$, based on the relative values of the charge levels

Deletion-error in rank modulated flash memories was introduced by Gabrys et al. [19]. During programming cells, deletions occur when the cells are corrupted and the charge levels cannot be read correctly. Since the information is stored in blocks of n cells based on relative charge levels, a deletion affects the information of the remaining cells, this suggests the need of deletion-correcting codes over permutations or multipermutations. Gabrys et al. constructed asymptotically optimal permutation codes that can correct a single deletion.

Our contribution. We study *burst-deletion correcting codes* for permutations and multipermutations. We design codes that can correct a burst of length exactly s or of length up to s , where $s \geq 2$. We prove the correctness of our constructions by

providing efficient error decoders that run in time linear in the length of permutations or multipermutations. We also introduce *burst-insertion correcting codes* for permutations and multipermutations for theoretical interest.

1.4 Duplication-Correcting Codes for DNA Storage

A *tandem duplication*, which is a special type of insertion, also known as *sticky insertion*, has been investigated recently due to the application in storing information in the DNA of living organism. In 2001, Lander et al. [39] published a draft sequence of the human genome and reported that more than 50% of the human genome consists of repeated substrings [39]. There are two types of common repeats: interspersed repeats and tandem repeats. Interspersed repeats are caused by transposons when a segment of DNA is copied and pasted into new positions of the genome. The other type, tandem repeats, are caused by slipped-strand mispairings, and they occur when a pattern of one or more nucleotides is repeated and the repetitions are adjacent to each other. For example, consider the word GTATAC. The substring TATA is a tandem repeat, and we say that GTATAC is generated from GTAC by a tandem duplication of length two. Tandem repeats are believed to be the cause of several genetic disorders [40, 41] and this motivated the study of tandem duplication mechanisms in a variety of contexts.

Formal languages. Leopold et al. [37, 38] defined the unbounded duplication language and the k -bounded duplication language to be the set of words generated by seed word under unlimited number of tandem duplications of unbounded length and tandem duplications of length up to k , respectively. They investigated certain decidability problems involving unbounded duplication languages and showed that all k -bounded duplication languages are context free. Furthermore, they showed that k -bounded duplication language is always regular for any binary seed and $k \geq 1$. On the other hand, the k -bounded duplication language is not regular for any square-free

seed word over an alphabet of at least three letters and $k \geq 4$. More recently, Jain et al. [45] completed this characterization and proved that k -bounded duplication languages are regular for $k \leq 3$.

Codes correcting tandem duplications. Motivated by applications that store data in living organisms [42–44], Jain et al [45] proposed the study of codes that correct tandem duplications to improve the reliability of data storage. They investigated various types of tandem duplications and provided optimal code construction in the case where duplication length is at most two. All constructions are based on *irreducible words*. To correct duplications of length three, they also use irreducible words to construct codes with sizes that are close to the upper bound. However, this technique does not work when the length of duplications are more than three.

Our contribution. We provide two efficient algorithms to encode/decode irreducible words. Since irreducible words play an important role in the design of codes correcting tandem duplications, our work provides first known efficient encoder/decoder for codes correcting tandem duplications. In addition, we give the first known result on codes that can correct tandem duplications of length at most k , for arbitrary $k \geq 4$.

1.5 Organization

Chapter 2 provides mathematical preliminaries including some basic notation and results on coding theory, combinatorics, algebra, graph theory that are used in this thesis.

Chapter 3 deals with the analysis and constructions of permutation codes that are capable of correcting a single burst of deletions/insertions. Our results include codes correcting a burst of length exactly s or of length up to s for some constant $s \geq 2$.

Chapter 4 provides constructions for multipermutation codes correcting a burst of deletions/insertions. We apply the coding technique to construct q -ary constant-composition codes correcting a burst of deletions/insertions. We also provide some

constructions when there is no restriction on the composition.

Chapter 5 investigates q -ary codes correcting tandem duplication errors. Other than code analysis and constructions, we also present efficient encoding/decoding algorithms for codes correcting tandem duplications of length up to three.

1.6 Bibliographical Notes

Some materials presented in this thesis have been presented at IEEE International Symposium on Information Theory (ISIT) and submitted to journals.

- The result on “permutation codes correcting a single burst of deletions” in Chapter 3 was present at the 2017 IEEE ISIT in Aachen, Germany.
- The result on “efficient encoding/decoding of irreducible words for codes correcting tandem duplications” in Chapter 5 was presented at the 2018 IEEE ISIT in Vail, USA.

Chapter 2

Preliminaries

2.1 Basic Notation

The set of integers, rational numbers, and real numbers are denoted as $\mathbb{Z}, \mathbb{Q}, \mathbb{R}$ respectively. We use $\mathbb{Z}_+, \mathbb{Q}_+, \mathbb{R}_+$ to denote the set of positive integer, rational, and real numbers, respectively. For a real number $x \in \mathbb{R}$, $\lfloor x \rfloor$ denotes the largest integer that is not greater than x , and $\lceil x \rceil$ denotes the smallest integer that is not smaller than x .

For two integers $a \leq b$, let $[a, b]$ denote the set $\{a, a+1, a+2, \dots, b\}$. For a positive integer n , we denote S_n as the set of all *permutations* on the set $[1, n]$. For example, when $n = 4$, $S_3 = \{123, 132, 213, 231, 312, 321\}$. In general, the size of S_n is $n!$.

Let Σ be an alphabet. For a positive integer n , we denote Σ^n as the set of vectors $\mathbf{v} = (v_1, v_2, \dots, v_n)$ with n components where its entries are in Σ , i.e $v_i \in \Sigma$ for $i \in [1, n]$. In addition, if $\Sigma = \{0, 1\}$, $\mathbf{v}$ is called a *binary vector*, and if $\Sigma = \{0, 1, \dots, q-1\}$ for some integer $q \geq 2$, $\mathbf{v}$ is called a *q-ary vector*.

For two vectors $\mathbf{u} = (u_1, u_2, \dots, u_m)$ and $\mathbf{v} = (v_1, v_2, \dots, v_n)$, the *concatenation* of $\mathbf{u}$ and $\mathbf{v}$ is the vector $\mathbf{u}||\mathbf{v} = (u_1, u_2, \dots, u_m, v_1, v_2, \dots, v_n)$.

For two sets M and N , we define the intersection, union, and set difference of M and N as $M \cap N = \{x | x \in M \text{ and } x \in N\}$, $M \cup N = \{x | x \in M \text{ or } x \in N\}$ and $M \setminus N = \{x \in M | x \notin N\}$, respectively.

2.2 Coding Theory

Let $\mathfrak{X}$ be a metric space with a distance function $d : \mathfrak{X}^2 \rightarrow \mathbb{R}$. A *code* $\mathcal{C}$ is a subset of $\mathfrak{X}$, and each element of $\mathcal{C}$ is called a *codeword*. A *q-ary code of length n* is a subset $\mathcal{C} \subset \{0, 1, \dots, q-1\}^n$. When $q = 2$, $\mathcal{C}$ is called a *binary code*. Similarly, a subset $\mathcal{C} \subset S_n$ is called a *permutation code of length n*. The *minimum distance* of the code $\mathcal{C}$ is the largest number d such that for all $c_1 \neq c_2, c_1, c_2 \in \mathcal{C}$, $d(c_1, c_2) \geq d$. Given $d \in \mathbb{Z}_+$ and a metric space $\mathfrak{X}$, the fundamental problem in coding theory is to construct the largest code $\mathcal{C}$ such that the minimum distance is at least d . Let $A(n, d)$ denote the size of the largest possible code. A code $\mathcal{C}$ is called *optimal* if and only if the size of $\mathcal{C}$, denoted by $|\mathcal{C}|$, is exactly $A(n, d)$.

The *rate* of a q -ary code $\mathcal{C}$ of length n is defined as $\text{Rate}(\mathcal{C}) = \frac{\log_q |\mathcal{C}|}{n}$. Suppose that we have an infinite family of codes $\{\mathcal{C}_n\}_{n=1}^\infty$ where $\mathcal{C}_n$ is a code of length n . Then the *asymptotic rate of the family* is $R(\mathcal{C}) = \lim_{n \rightarrow \infty} \frac{\log_q |\mathcal{C}_n|}{n}$. The notation “log” without the base refers to logarithm of base two. When $R(\mathcal{C}) = 1$, we say that the family of codes is asymptotically optimal.

The *Hamming distance* between two codewords $\mathbf{u} = (u_1, u_2, \dots, u_n)$ and $\mathbf{v} = (v_1, v_2, \dots, v_n)$ in $\mathcal{C}$ is denoted by $d_H(\mathbf{u}, \mathbf{v})$, and $d_H(\mathbf{u}, \mathbf{v}) = |\{i | \mathbf{u}_i \neq \mathbf{v}_i\}|$.

Sphere-packing bound. For an element $\mathbf{x} \in \mathfrak{X}$, a ball of radius r centered at $\mathbf{x}$ is defined as follows.

$$\mathfrak{B}_r(\mathbf{x}) = \{\mathbf{y} \in \mathfrak{X} : d(\mathbf{x}, \mathbf{y}) \leq r\}$$

If $|\mathfrak{B}_r(\mathbf{x})| = B_r$, which is independent of $\mathbf{x}$ and $\mathcal{C} \in \mathfrak{X}$ is a code with minimum distance d then

$$|\mathcal{C}| \leq \frac{|\mathfrak{X}|}{B_t},$$

where $t = \lfloor (d-1)/2 \rfloor$. For the binary Hamming space, we have $B_r = \sum_{i=0}^r \binom{n}{i}$.

Therefore

$$|\mathcal{C}| \leq \frac{2^n}{\sum_{i=0}^t \binom{n}{i}}.$$

Similarly, for q -ary Hamming space, we have $B_r = \sum_{i=0}^r \binom{n}{i} (q-1)^i$. Consequently

$$|\mathcal{C}| \leq \frac{q^n}{\sum_{i=0}^t \binom{n}{i} (q-1)^i}.$$

2.3 Analysis

The following classical notations will be used throughout the thesis for the purpose of running time analysis. Let $f, g : \mathbb{R} \rightarrow \mathbb{R}_+$ be two positive real-valued functions.

- $f(n) = O(g(n))$: if there are positive numbers c and N such that $f(n) \leq cg(n)$ for all $n \geq N$,
- $f(n) = \Omega(g(n))$: if there are positive numbers c and N such that $f(n) \geq cg(n)$ for all $n \geq N$,
- $f(n) = \Theta(g(n))$: if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$,
- $f(n) = o(g(n))$: if $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$.

2.4 Number Theory

There are several codes constructed based on *congruent class*. Let $a, b \in \mathbb{Z}, n \in \mathbb{Z}_+$. Then a is congruent to b modulo n if n divides $a - b$, and we write $a \equiv b \pmod{n}$. The congruent class of a modulo n , denoted by $[a]_n$ is the set of all integers that are congruent to a modulo n , mathematically

$$[a]_n = \{z \in \mathbb{Z} : z \equiv a \pmod{n}\}$$

Two congruence classes modulo n are either disjoint or identical.

Chapter 3

Permutation Codes

In this chapter, we construct *burst deletion/insertion-correcting codes* for permutations. The motivation of this work is the application of rank modulation scheme for flash memories. Deletions and insertions in permutations have been classified into two different types in [19].

3.1 Classification of Deletions/Insertions in Permutations

In rank modulation scheme, the information is stored as blocks of n cells, which represents a permutation of length n based on the relative values of the charge levels. Gabrys et al. [19] classified deletions into two following types, according to how the errors affect the information of the remaining cells.

- (i) *Stable deletion*: In such deletions, the remaining symbols stay unchanged. Consequently, the value of deleted symbols are known. The aim of code design is to find the locations of deleted symbols.
- (ii) *Unstable deletion*: In such deletions, only the relative values of the remaining symbols are known. The received vector is a new permutation of length $n - s$, where s is the number of deletions.

Example 3.1.1. Suppose permutation $(5, 3, 6, 7, 8, 4, \textcolor{red}{2}, 1, 9)$ experiences a deletion of symbol two. In a stable deletion, the resulting vector is $(5, 3, 6, 7, 8, 4, 1, 9)$, whereas in an unstable deletion, the resulting vector is $(4, 2, 5, 6, 7, 3, 1, 8)$, which is a new permutation of length 8.

We now define a burst of stable and unstable deletions in permutations as well as code correcting a burst of stable and unstable deletions.

Definition 3.1.2. Assume that $\pi = (\pi_1, \pi_2, \dots, \pi_n) \in \mathcal{S}_n$ and $I = \{i+1, i+2, \dots, i+s\} \subseteq [1, n]$ is a set of size s . We say the permutation π suffers a *burst of s stable deletions* in I , resulting in the vector $(\pi_1, \dots, \pi_i, \pi_{i+s+1}, \dots, \pi_n)$. We use π_I to denote this resulting vector. In general, we use $\mathcal{B}_s(\pi)$ to denote the set of all vectors of length $n - s$ received as a result of a burst of s stable deletions in π .

Example 3.1.3. Let $\pi = (7, 1, 2, \textcolor{red}{3}, \textcolor{red}{5}, 6, 4)$ and $I = \{4, 5\}$. If π experiences two stable deletions in I , then the resulting vector is $(7, 1, 2, 6, 4)$.

Definition 3.1.4. A code $\mathcal{C} \subseteq \mathcal{S}_n$ is called an *s -BSD permutation code* if it can correct a burst of exactly s stable deletions, or an *$\leq s$ -BSD permutation code* if it can correct a burst of up to s stable deletions.

To define unstable deletion in permutations, we introduce some notations. For a permutation $\pi = (\pi_1, \pi_2, \dots, \pi_n) \in \mathcal{S}_n$ and a set of positions $I \subseteq [1, n]$, define $\pi(I) = \{\pi_i : i \in I\}$. Let X be a set of numbers. For a number a , we define $a(X) = a - |\{x \in X : x < a\}|$.

Example 3.1.5. if $\pi = (5, 1, 4, 2, 6, 3)$ and $I = \{3, 4\}$, then $\pi(I) = \{2, 4\}$, $5(\pi(I)) = 5 - 2 = 3$, $1(\pi(I)) = 1 - 0 = 1$, and $6(\pi(I)) = 6 - 2 = 4$.

Definition 3.1.6. Assume that $\pi = (\pi_1, \pi_2, \dots, \pi_n) \in \mathcal{S}_n$ and $I = \{i+1, \dots, i+s\} \subseteq [1, n]$ is a set of size s . We say that π suffers a *burst of s unstable deletions* in I , resulting in the permutation $(\tilde{\pi}_1, \dots, \tilde{\pi}_{n-s}) \in \mathcal{S}_{n-s}$, where $\tilde{\pi}_j = \pi_j(\pi(I))$ for $1 \leq j \leq i$

and $\tilde{\pi}_j = \pi_{j+s}(\pi(I))$ for $i + 1 \leq j \leq n - s$. We use $\pi_{\downarrow, I}$ to denote this resulting permutation. In general, we use $\mathcal{B}_{\downarrow, s}(\pi)$ to denote the set of all permutations in $\mathcal{S}_{n-s}$ received as a result of a burst of s unstable deletions in π .

As above example, $\pi = (7, 1, 2, \textcolor{red}{3}, \textcolor{red}{5}, 6, 4)$ and $I = \{4, 5\}$. If π experiences two unstable burst deletions in I , then the resulting vector is a new permutation of length five, which is $\tilde{\pi} = (5, 1, 2, 4, 3)$.

Definition 3.1.7. A code $\mathcal{C} \subseteq S_n$ is called an *s-BUD permutation code* if it can correct a burst of exactly s unstable deletions, or an *$\leq s$ -BUD permutation code* if it can correct a burst of up to s unstable deletions.

We now define insertions in permutations.

- (i) *Stable Insertion:* Similar to the case of stable deletions, when stable insertions occur, the absolute values of the remaining symbols stay unchanged. Consequently, the value of inserted symbols are known and the resulting vector is a multipermutation. The aim of code design is to find the locations of inserted symbols.
- (ii) *Unstable Insertion:* Similar to the case of unstable deletions, when unstable insertions occur, we only know the relative values of the remaining symbols. The resulting vector is a new permutation of length $n + s$, where s is the number of insertions. When an insertion occurs, all symbols that are smaller than the inserted symbol stay unchanged whereas the other symbols increase their values by one.

Example 3.1.8. Suppose there is an insertion of symbol two occurring at the end in the permutation $(5, 3, 2, 1, 4)$. In a stable insertion, it gives the vector $(5, 3, 2, 1, 4, 2)$, whereas in an unstable insertion, it gives the vector $(6, 4, 2, 1, 5, 3)$.

A burst of stable insertions is similar to a burst of stable deletions where the value of deleted/inserted symbols are known and we only need to find the location of errors.

We now explain how a burst of unstable insertions occurs. Suppose that there is a burst of two unstable insertions, i.e “31” is inserted at the end of permutation $\pi = (5, 3, 2, 1, 4) \in \mathcal{S}_5$. Firstly, the smaller symbol, symbol one, is inserted at the end of π , and the resulting permutation is $\pi' = (6, 4, 3, 2, 5, \mathbf{1})$. Next, symbol three is inserted and the final resulting permutation is $\pi'' = (7, 5, 4, 2, 6, \mathbf{3}, \mathbf{1})$. In this case, the inserted 3 becomes the new 3 and the old 3 now moves up to a 4.

Definition 3.1.9. A code $\mathcal{C} \subseteq \mathcal{S}_n$ is called an *s-BSI permutation code* if it can correct a burst of exactly s stable insertions, or an *$\leq s$ -BSI permutation code* if it can correct a burst of up to s stable insertions. We define *s-BUI permutation code*, *$\leq s$ -BUI permutation code* similarly where BUI stands for “burst of unstable insertions”.

Gabrys et al. [19] proved the equivalence of *s-BUD* permutation codes and *s-BUI* permutation codes. Specifically, they showed that “for all $s \geq 1$, $\mathcal{C}$ is an *s-BUD* permutation code if and only if it is also an *s-BUI* permutation code”. In this chapter, we also prove the equivalence of *s-BSD* permutation codes and *s-BSI* permutation codes. We then construct *s-BSD* permutation codes and use these results for *s-BSI* permutation codes.

In the next subsection, we summarize the previous results on single error in permutations. The references are the works of Gabrys et al. [19] and Levenshtein [18]. One may use *interleaving technique* to construct codes against burst errors of length s as follows. For a codeword $\mathbf{x}$ of length n , we view it as an $s \times n/s$ array. A burst of s deletions/insertions simply deletes/inserts exactly one symbol in a row. Therefore, if we use each row as a code that can correct a single deletion/insertion, then the interleaved code is capable of correcting a burst of length s . We formally describe this trivial construction in the next section. This construction is simple and the decoding algorithm is simple, however, the redundancy is very large.

3.2 Previous Results

3.2.1 Single Stable Deletion

Given $a \in \mathbb{Z}_{n+1}$, let

$$\mathcal{C}_a(n) = \left\{ \mathbf{x} \in \{0, 1\}^n, \mathbf{x} = (x_1, x_2, \dots, x_n) : \sum_{i=1}^n ix_i \equiv a \pmod{(n+1)} \right\}.$$

The binary codes $\mathcal{C}_a(n)$ are known as the *Varshamov-Tenengolts (VT) codes* [29], which are capable of correcting a single deletion or insertion. Since the union of these codes over $a \in \mathbb{Z}_{n+1}$ is the set of all binary words of length n , which has size 2^n , the largest code should have size at least $2^n/(n+1)$.

As in [30], Tenengolts defined the *signature* of a vector $\mathbf{x} = (x_1, x_2, \dots, x_n)$ to be a binary vector $\alpha(\mathbf{x}) = (\alpha(x_1), \dots, \alpha(x_{n-1}))$ of length $n-1$, where $\alpha(x_i) = 1$ if $x_{i+1} \geq x_i$, and 0 otherwise, for all $i \in [1, n-1]$. For example, if $\mathbf{x} = (3, 3, 4, 2, 1, 3)$, then $\alpha(\mathbf{x}) = (1, 1, 0, 0, 1)$. Given $a \in \mathbb{Z}_n$, Levenshtein [18] showed that

$$\mathcal{P}_a(n) = \{\sigma \in S_n : \alpha(\sigma) \in \mathcal{C}_a(n-1)\}$$

is a permutation code that is capable of correcting a single stable deletion. The union of these codes over $a \in \mathbb{Z}_n$ is S_n , the largest code should have size at least $n!/n = (n-1)!$.

Tenengolts [30] also generalized the binary VT codes to nonbinary ones. For any $a \in \mathbb{Z}_n$ and $b \in \mathbb{Z}_q$, let

$$\mathcal{C}_{a,b}(n; q) = \left\{ \mathbf{x} \in [1, q]^n : \alpha(\mathbf{x}) \in \mathcal{C}_a(n-1) \text{ and } \sum_{i=1}^n x_i \equiv b \pmod{q} \right\}. \quad (3.1)$$

The codes $\mathcal{C}_{a,b}(n; q)$ are capable of correcting a single deletion. The union of these codes over $a \in \mathbb{Z}_n, b \in \mathbb{Z}_q$ is the set of all q -ary words of length n , which has size q^n . Therefore, there exists a code whose size is at least q^{n-1}/n .

3.2.2 Single Unstable Deletion

For a permutation $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_n) \in S_n$, its *inverse* is the permutation $\sigma^{-1} = (\sigma_1^{-1}, \sigma_2^{-1}, \dots, \sigma_n^{-1})$, where σ_i^{-1} is the location of i in σ . Given $a, b \in \mathbb{Z}_n$, let

$$\mathcal{P}_{a,b}(n) = \left\{ \sigma \in S_n : \alpha(\sigma) \in \mathcal{C}_a(n-1) \text{ and } \alpha(\sigma^{-1}) \in \mathcal{C}_b(n-1) \right\}.$$

The construction above is presented by Gabrys et al. [19]. They showed that, for each choice of $a, b \in \mathbb{Z}_n$, $\mathcal{P}_{a,b}(n)$ is a permutation code that can correct a single unstable deletion. Since the union of these codes over $a, b \in \mathbb{Z}_n$ is S_n , which has size $n!$, there exists a code whose size is at least $n!/n^2$. They also showed that the size is asymptotically optimal.

3.3 Permutation Codes Correcting a Burst of Stable Deletions

3.3.1 An Upper Bound

Theorem 3.3.1. Given $n, s \in \mathbb{Z}_+, n > s$. A code $\mathcal{C} \in \mathcal{S}_n$ is an s -BSD permutation code. We have

$$|\mathcal{C}| \leq \frac{n!}{s!(n-s+1)}.$$

Proof. For $\pi \in \mathcal{S}_n$, let $\mathcal{D}_s(\pi)$ be the set of all vectors of length $n-s$ that can be received from π when a burst of exactly s stable deletions occurs. We have $|\mathcal{D}_s(\pi)| = n-s+1$.

Let $\mathcal{D}_s(\mathcal{S}_n) = \bigcup_{\pi \in \mathcal{S}_n} \mathcal{D}_s(\pi)$. It is easy to see that $|\mathcal{D}_s(\mathcal{S}_n)| = n!/s!$ since $\mathcal{D}_s(\mathcal{S}_n)$ is the set of all sequences containing $n-s$ distinct symbols over an alphabet of size n .

A code $\mathcal{C}$ is an s -BSD permutation code if and only if for distinct $\pi, \pi' \in \mathcal{C}$, we have $\mathcal{D}_s(\pi) \cap \mathcal{D}_s(\pi') = \emptyset$. Therefore, $|\mathcal{C}| \leq \frac{n!}{s!(n-s+1)}$. $\square$

3.3.2 Code Constructions for Burst of length s

When $n = st$ for some positive integers s and t , we rewrite a permutation $\pi = (\pi_1, \pi_2, \dots, \pi_n)$ as an array

$$\begin{bmatrix} \pi_1 & \pi_{s+1} & \cdots & \pi_{(j-1)s+1} & \cdots & \pi_{(t-1)s+1} \\ \pi_2 & \pi_{s+2} & \cdots & \pi_{(j-1)s+2} & \cdots & \pi_{(t-1)s+2} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \pi_s & \pi_{2s} & \cdots & \pi_{js} & \cdots & \pi_n \end{bmatrix}.$$

Hence, we may call this array as an $s \times t$ permutation and we denote the i th row by

$$\pi_{(s,i)} = (\pi_i, \pi_{s+i}, \dots, \pi_{n-s+i}), \text{ and}$$

the j th column by

$$\pi^{(s,j)} = (\pi_{(j-1)s+1}, \pi_{(j-1)s+2}, \dots, \pi_{js}).$$

Observation. A burst of exactly s deletions deletes exactly one symbol from each row and spans at most two adjacent columns. In stable deletion, the values of other symbols stay unchanged. Therefore, the decoder always knows the set of deleted symbols. It is only required to determine the deleted symbols' positions. The first trivial construction is as follows.

Construction 3.3.2. Given $n, s, t \in \mathbb{Z}_+, n = st$, let $\mathbf{a} = (a_1, a_2, \dots, a_s)$, and $\mathbf{b} = (b_1, b_2, \dots, b_s)$, where $a_i, b_i \in \mathbb{Z}_n, 1 \leq i \leq s$. Let $\mathcal{C}_s(n; \mathbf{a}, \mathbf{b})$ be the set of all permutations $\pi \in \mathcal{S}_n$ such that the i th row of π , $\pi_{(s,i)} \in \mathcal{C}_{a_i, b_i}(t; n)$.

Recall that $\mathcal{C}_{a_i, b_i}(t; n)$ is a code of length t over alphabet of size n that can correct a single stable deletion (as introduced in 3.1). Each row is therefore capable of correcting a single stable deletion. Consequently, permutation π can correct a burst of exactly s stable deletions.

Theorem 3.3.3. The code $\mathcal{C}_s(n; \mathbf{a}, \mathbf{b})$ from Construction 3.3.2 is an s -BSD permutation code. There exists $\mathbf{a}, \mathbf{b}$ such that the size of $\mathcal{C}_s(n; \mathbf{a}, \mathbf{b})$ is at least $n!/n^{2s}$.

This construction is simple, however, the redundancy is very large when the value of n gets larger. We now present our main result on this section for s -BSD permutation code and the code's size is improved from $\Theta(n!/n^{2s})$ to $\Theta(n!/n^2)$. Our main idea are based on the *permutation projection* function $\mathbf{p}$ and the *permutation rank* function μ . In addition, the technique can be efficiently extended to multipermutation codes, q -ary constant composition codes (q -ary balanced codes) which is shown later in Chapter 4. In addition, it is also applicable for codes correcting insertions.

Definition 3.3.4. Let $\mathbf{x} = (x_1, x_2, \dots, x_m)$ be an integer vector with different components, i.e. $x_i \neq x_j$ for distinct $i, j \in [1, m]$. We define the *permutation projection* of $\mathbf{x}$ as $\mathbf{p}(\mathbf{x})$, which is a permutation in $\mathcal{S}_m$ by the following rule:

$$\mathbf{p}(\mathbf{x})_i = |\{j : x_j \leq x_i, 1 \leq j \leq m\}|.$$

Observation. $\mathbf{p}(\mathbf{x})$ transforms a vector $\mathbf{x}$ to a permutation based on the relative values of its components. For example, if $\mathbf{x} = (1, 8, 6, 5, 4)$, then $\mathbf{p}(\mathbf{x}) = (1, 5, 4, 3, 2)$.

Definition 3.3.5. Let $\ell : \mathcal{S}_m \rightarrow [1, m!]$ such that $\ell(\pi)$ is the lexicographic rank of π in $\mathcal{S}_m$. For a vector $\mathbf{x}$ of length m , we define the *permutation rank* function $\mu : \mathbb{Z}^m \rightarrow [1, m!]$ as follows.

$$\mu(\mathbf{x}) = \ell(\mathbf{p}(\mathbf{x})).$$

Observation. From $\mu(\mathbf{x})$ and the set $\{x_i : 1 \leq i \leq m\}$, we can determine the sequence $\mathbf{x}$. For example, if $m = 5$ and $\mu(\mathbf{x}) = 2$, then $\mathbf{p}(\mathbf{x}) = (1, 2, 3, 5, 4)$. Furthermore, if $\{x_1, x_2, x_3, x_4, x_5\} = \{10, 30, 80, 50, 100\}$, then $\mathbf{x} = (10, 30, 50, 100, 80)$. Recall that, for two vectors $\mathbf{x} = (x_1, x_2, \dots, x_m)$ and $\mathbf{y} = (y_1, y_2, \dots, y_n)$, the concatenation of $\mathbf{x}$ and $\mathbf{y}$ is the vector $\mathbf{x}||\mathbf{y} = (x_1, x_2, \dots, x_m, y_1, y_2, \dots, y_n)$. We now present our construction for s -BSD permutation codes.

Construction 3.3.6. Given $n, s, t \in \mathbb{Z}_+, n = st, t$ is even. Let $a \in \mathbb{Z}_t, b \in \mathbb{Z}_n$, and $c, d \in \mathbb{Z}_{(2s)!}$. We construct a code $\mathcal{C}_s(n; a, b, c, d)$ containing all permutations $\pi \in \mathcal{S}_n$, those satisfy the following constraints:

(i) **First row constraint:** $\pi_{(s,1)}$ is a codeword of nonbinary VT-code (as defined 3.2.1), that is, $\pi_{(s,1)} \in \mathcal{C}_{a,b}(t; n)$.

(ii) **Concatenation of adjacent columns constraint:**

$$\begin{aligned} (1) \quad & \sum_{i=1}^{t/2} \mu(\pi^{(s,2i-1)} || \pi^{(s,2i)}) \equiv c \pmod{(2s)!}, \\ (2) \quad & \sum_{i=1}^{t/2-1} \mu(\pi^{(s,2i)} || \pi^{(s,2i+1)}) \equiv d \pmod{(2s)!}. \end{aligned}$$

Theorem 3.3.7. The code $\mathcal{C}_s(n; a, b, c, d)$ from Construction 3.3.6 is an s -BSD permutation code over $\mathcal{S}_n$.

Proof. We prove the correctness of this theorem by providing a decoding algorithm for $\mathcal{C}_s(n; a, b, c, d)$ that can correct a burst of exactly s stable deletions. Suppose a burst of s stable deletions occurs in a codeword $\pi = (\pi_1, \pi_2, \dots, \pi_n)$, giving $\tilde{\pi} = (\tilde{\pi}_1, \tilde{\pi}_2, \dots, \tilde{\pi}_{(t-1)s})$. Our decoding algorithm has two steps.

Step 1: Recover the first row. We first select those symbols in the first row of the received word $\tilde{\pi}$, let $\zeta = (\tilde{\pi}_1, \tilde{\pi}_{s+1}, \dots, \tilde{\pi}_{(t-2)s+1})$. In fact, ζ is the resulting vector from a deletion in $\pi_{(s,1)}$. Since $\pi_{(s,1)} \in \mathcal{C}_{a,b}(t; n)$, we can recover $\pi_{(s,1)}$ from ζ . Moreover, it also allows us to identify the location of the deleted symbol since all components of $\pi_{(s,1)}$ are distinct. Suppose the deletion in the first row of π occurred at position $(i-1)s+1$, in other words, it is the first symbol in the i th column of π .

Step 2: Recover the burst of deletions.

Case 1. If $2 \leq i \leq t-1$. The burst of s deletions occurs in two consecutive columns of π , which are the $(i-1)$ th column and the i th column. We first identify the symbols at other columns and their positions in the original codeword as follows. Let $\gamma_j = (\tilde{\pi}_{(j-1)s+1}, \tilde{\pi}_{(j-1)s+2}, \dots, \tilde{\pi}_{js})$, which is the j th column of the received word $\tilde{\pi}$. We then have

(i) $\pi^{(s,j)} = \gamma_j$ for $1 \leq j < i - 1$, and

(ii) $\pi^{(s,j)} = \gamma_{j-1}$ for $i + 1 \leq j \leq t$,

Our final work is to recover two columns $\pi^{(s,i-1)}$ and $\pi^{(s,i)}$. Firstly, we can deduce the set of symbols appearing in these two columns based on those recovered columns. We now compute $\mu(\pi^{(s,i-1)} || \pi^{(s,i)})$ to recover these two columns.

When i is even,

$$\mu(\pi^{(s,i-1)} || \pi^{(s,i)}) \equiv c - \sum_{j \in [1, t/2] \setminus \{i/2\}} \mu(\pi^{(s,2j-1)} || \pi^{(s,2j)}) \pmod{(2s)!}, \text{ and}$$

when i is odd,

$$\mu(\pi^{(s,i-1)} || \pi^{(s,i)}) \equiv d - \sum_{j \in [1, t/2-1] \setminus \{(i-1)/2\}} \mu(\pi^{(s,2j)} || \pi^{(s,2j+1)}) \pmod{(2s)!}.$$

Case 2. If $i = 1$ or $i = t$. If $i = 1$, the burst of s deletions spans the whole first column. We then compute $\mu(\pi^{(s,1)} || \pi^{(s,2)})$ to recover this column. On the other hand, if $i = t$, the burst spans at most two columns, which are the $(t-1)$ th column and the t th column. Similarly, we just need to compute $\mu(\pi^{(s,t-1)} || \pi^{(s,t)})$ to recover these two columns.

When $i = 1$,

$$\mu(\pi^{(s,1)} || \pi^{(s,2)}) \equiv c - \sum_{j \in [2, t/2]} \mu(\pi^{(s,2j-1)} || \pi^{(s,2j)}) \pmod{(2s)!}, \text{ and}$$

When $i = t$,

$$\mu(\pi^{(s,t-1)} || \pi^{(s,t)}) \equiv c - \sum_{j \in [1, t/2-1]} \mu(\pi^{(s,2j-1)} || \pi^{(s,2j)}) \pmod{(2s)!}.$$

□

Corollary 3.3.8. There exists an s -BSD permutation code of size at least $\frac{s}{((2s)!)^2} \cdot \frac{n!}{n^2}$.

Proof. We have $\cup_{a \in \mathbb{Z}_t, b \in \mathbb{Z}_n, c, d \in \mathbb{Z}_{(2s)!}} \mathcal{C}_s(n; a, b, c, d) = \mathcal{S}_n$, where $t = \frac{n}{s}$. There are totally $t \times n \times (2s!)^2 = \frac{n^2(2s!)^2}{s}$ different classes, hence there exists a code of size at least

$$\frac{n!}{\frac{n^2(2s!)^2}{s}} = \frac{s}{((2s)!)^2} \cdot \frac{n!}{n^2}.$$

□

When s is fixed, our construction forms a family of s -BSD permutation codes whose size is at least $\Theta(n!/n^2)$ while the upper bound is $\Theta(n!/n)$. This size is much larger than size of codes in Construction 3.3.2. We now analyze the running time for the error decoder. Throughout this thesis, the running time is measured in integer operations.

Step 1: Recover the first row. This can be done efficiently in time linear in the length of the first row, which is $n/s = \Theta(n)$ (as proved in [30]).

Step 2: Recover the burst of deletions. The complexity of addition/subtraction over $\Theta(n)$ numbers is $\Theta(n)$.

In conclusion, the error decoder runs in time linear in the length of permutation.

3.3.3 Permutation Code Constructions for Burst of length up to s

In this subsection, we design $\leq s$ -BSD permutation codes. Since deletions occur randomly, the length of a burst might be uncertain. Therefore, codes correcting a burst of length up to s are more useful in the practical model than codes a burst of length exactly s . In general, an s -BSD code does not automatically yield an $\leq s$ -BSD code, even though it might seem that fewer deletions are easier to handle. We may delete some extra symbols at the decoder to make sure there are exactly s deletions, but those deletions might not be at the adjacent positions, therefore, we can not use the

s -BSD decoder.

In general, the idea is taking the intersection of a family of $\mathcal{C}_i(n; a_i, b_i, c_i, d_i)$, where $2 \leq i \leq s$, together with $\mathcal{P}_{a_1}(n)$ (defined in Subsection 3.2.1). It is easy to see that such a code has size at least

$$\frac{s!n!}{\prod_{i=2}^s ((2i)!)^2 n^{2s-1}}.$$

We now show that we can do better, with the assumption that $i \mid n$ for $1 \leq i \leq s$, and $n = 2ts$ for some even integer t .

Construction 3.3.9. Let $\mathbf{a} = (a_1, a_2, \dots, a_s)$, where $a_i \in \mathbb{Z}_{n/i}$, for $i = 1, 2, \dots, s$, $\mathbf{b} = (b_2, \dots, b_s)$, where $b_j \in \mathbb{Z}_n$, for $j = 2, 3, \dots, s$, and $c, d \in \mathbb{Z}_{(4s)!}$. We construct a code $\mathcal{C}_s(n; \mathbf{a}, \mathbf{b}, c, d)$ containing all permutations $\pi \in \mathcal{S}_n$, those satisfy the following constraints:

(i) **First row constraint:** When we write π as an $i \times (n/i)$ permutation,

(1) If $i = 1$, the first row is actually the whole codeword π . We require

$$\pi \in \mathcal{P}_{a_1}(n) \text{ (defined in 3.2.1).}$$

(2) For $2 \leq i \leq s$, the first row is $\pi_{(i,1)}$ of length n/i . We require

$$\pi_{(i,1)} \in \mathcal{C}_{a_i, b_i}(n/i; n).$$

(ii) **Concatenation of adjacent columns constraint:** When we write π as a $2s \times t$ permutation, we require

(1) $\sum_{i=1}^{t/2} \mu(\pi^{(2s, 2i-1)} || \pi^{(2s, 2i)}) \equiv c \pmod{(4s)!}$, and

(2) $\sum_{i=1}^{t/2-1} \mu(\pi^{(2s, 2i)} || \pi^{(2s, 2i+1)}) \equiv d \pmod{(4s)!}$.

Theorem 3.3.10. The code $\mathcal{C}_s(n; \mathbf{a}, \mathbf{b}, c, d)$ from Construction 3.3.9 is an $\leq s$ -BSD permutation code.

Proof. We prove the correctness of the theorem by giving an efficient error decoder for $\mathcal{C}_s(n; \mathbf{a}, \mathbf{b}, c, d)$ when there is a burst of up to s deletions.

Suppose there are ℓ stable deletions, $\ell \leq s$, that occur in a codeword $\pi = (\pi_1, \pi_2, \dots, \pi_n) \in \mathcal{C}_s(n; \mathbf{a}, \mathbf{b}, c, d)$. The received word is $\tilde{\pi} = (\tilde{\pi}_1, \tilde{\pi}_2, \dots, \tilde{\pi}_{n-\ell})$. Our decoding algorithm has three steps.

Step 1: Determine the number of deletions. From the length of received word $\tilde{\pi}$, we can determine the value of ℓ .

Step 2: Recover the first row. We now view π as an $\ell \times (n/\ell)$ permutation and recover the first row of π , which is $\pi_{(\ell,1)}$, as follows.

- If $\ell = 1$, there is only one deletion in π , we can recover π efficiently as $\pi \in \mathcal{P}_{a_1}(n)$.
- If $\ell \geq 2$, $\pi_{(\ell,1)}$ can be recovered efficiently as in the proof of Theorem 3.3.7.

Step 3: Recover the burst of deletions. Since the first row is recovered, we also know the location of the deletion. Suppose $\pi_{i\ell+1}$ is deleted, in other words, this is the first symbol in the i th column. Since the burst spans at most two columns, the $(i-1)$ th and i th, we can recover those symbols in other columns.

- (i) $\pi_j = \tilde{\pi}_j$, for $1 \leq j \leq i\ell - \ell + 1$; and
- (ii) $\pi_j = \tilde{\pi}_{j-\ell}$, for $i\ell + \ell + 1 \leq j \leq n$.

Our final work is to recover two remaining columns, the $(i-1)$ th and i th column, in other words, we just need to determine the sequence $\pi_{(i-1)\ell+2}, \pi_{(i-1)\ell+3}, \dots, \pi_{(i+1)\ell}$ of length $2\ell - 1$. We now view π as a $2s \times (n/2s)$ permutation. Then the sequence to be determined spans at most two adjacent columns of π and can be recovered as in the proof of Theorem 3.3.7. Notice that if $i = 0$, the burst spans within the first column, we can also recover it by computing $\mu(\pi^{(2s,0)} || \pi^{(2s,1)})$. $\square$

Corollary 3.3.11. There exists an $\leq s$ -BSD permutation code of size at least $\frac{s!n!}{((4s)!)^2 n^{2s-1}}$.

Proof. We have $\bigcup_{a_i \in \mathbb{Z}_{n/i}, b_j \in \mathbb{Z}_n, c, d \in \mathbb{Z}_{(4s)!}} \mathcal{C}_s(n; \mathbf{a}, \mathbf{b}, c, d) = \mathcal{S}_n$. Similar to Corollary 3.3.8, there exists a code $\mathcal{C}_s(n; \mathbf{a}, \mathbf{b}, c, d)$ whose size is at least as large as the average. $\square$

3.3.4 A Better Construction for $s = 2$

When $s = 2$, from Corollary 3.3.8 and Corollary 3.3.11, the code from Construction 3.3.6 has size at least $1/288 \times n!/n^2$ and code from Construction 3.3.9 has size at least $1/812851200 \times n!/n^3$. In this subsection, we present another construction that can achieve much better results as $n!/n^2$ for 2-BSD permutation codes and $n!/n^3$ for ≤ 2 -BSD permutation codes.

Definition 3.3.12. For a vector $\mathbf{x} = (x_1, x_2, \dots, x_n)$, we define the *inversion fuction* $\beta(\mathbf{x})$ as follows.

$$\beta(\mathbf{x}) = |\{(x_i, x_j), i < j : x_i > x_j\}|$$

Example 3.3.13. If $\mathbf{x} = (1, 3, 3, 2)$ and $\mathbf{y} = (3, 2, 4, 1)$, We have $\beta(\mathbf{x}) = 2, \beta(\mathbf{y}) = 4$.

Recall that when $s = 2$ and $n = 2t$ for some integer t , a permutation $\pi = (\pi_1, \pi_2, \dots, \pi_n)$ is also written as the array

$$\begin{bmatrix} \pi_1 & \pi_3 & \cdots & \pi_{2i-1} & \cdots & \pi_{n-1} \\ \pi_2 & \pi_4 & \cdots & \pi_{2i} & \cdots & \pi_n \end{bmatrix}.$$

Construction 3.3.14. Let $a \in \mathbb{Z}_t$, $b \in \mathbb{Z}_n$, and $c \in \mathbb{Z}_2$. We construct a code $\mathcal{C}_2(n; a, b, c)$ containing all permutations $\pi \in \mathcal{S}_n$, those satisfy the following constraints:

- (i) **First row constraint:** The first row of π , $\pi_{(2,1)}$ is a codeword of nonbinary VT-code, that is, $\pi_{(2,1)} \in \mathcal{C}_{a,b}(t; n)$.
- (ii) **Second row constraint:** The second row of π , $\pi_{(2,2)} = (\pi_2, \pi_4, \dots, \pi_n)$, satisfies

$$\beta(\sigma_{(2,2)}) \equiv c \pmod{2}$$

Theorem 3.3.15. $\mathcal{C}_2(n; a, b, c)$ is a 2-BSD permutation code over $\mathcal{S}_n$.

Proof. We prove the correctness of Theorem 3.3.15 by giving an error decoder for $\mathcal{C}_2(n; a, b, c)$ when there are two adjacent stable deletions.

Suppose that two deletions occur in a codeword $\pi = (\pi_1, \pi_2, \dots, \pi_n) \in \mathcal{C}_2(n; a, b, c)$, and the resulting vector is $\tilde{\pi} = (\tilde{\pi}_1, \tilde{\pi}_2, \dots, \tilde{\pi}_{2t-2})$. Our decoding algorithm has three steps.

Step 1: Identify two deleted symbols. From the received vector, $\tilde{\pi}$, we first identify two deleted symbols.

Step 2: Recover the first row. We recover the first row of π by the same method in the proof of Theorem 3.3.7 since $\pi_{(2,1)} \in \mathcal{C}_{a,b}(t; n)$.

Step 3: Recover the second row. Suppose the deleted symbol in the second row is x . Assume that the deletion in the first row occurred at position $(2j_1 - 1)$, in other words, π_{2j_1-1} was deleted. We then have only two possible sequences for the second row of original permutation.

$$(i) \quad \pi_{2,2} = A = (\tilde{\pi}_2, \dots, \tilde{\pi}_{2j_1-4}, \tilde{\pi}_{2j_1-2}, x, \tilde{\pi}_{2j_1}, \dots, \tilde{\pi}_{2t-2}), \text{ or}$$

$$(ii) \quad \pi_{2,2} = B = (\tilde{\pi}_2, \dots, \tilde{\pi}_{2j_1-4}, x, \tilde{\pi}_{2j_1-2}, \tilde{\pi}_{2j_1}, \dots, \tilde{\pi}_{2t-2}).$$

The second constraint $\beta(\pi_{(2,2)}) \equiv c \pmod{2}$ helps us distinguish these two sequences, since the values of $\beta(A)$ and $\beta(B)$ differ by one. $\square$

Corollary 3.3.16. There exists a 2-BSD permutation code of size at least $\frac{n!}{n^2}$.

Proof. We have $\bigcup_{a \in \mathbb{Z}_t, b \in \mathbb{Z}_n, c \in \mathbb{Z}_2} \mathcal{C}_2(n; a, b, c) = \mathcal{S}_n$, where $t = n/2$. There are $t \times n \times 2 = n^2$ different classes. Consequently, there exists a code of size least $n!/n^2$. $\square$

To construct an ≤ 2 -SBD permutation code, we take the intersection of $\mathcal{P}_a(n)$ (defined in subsection 3.2.1) and the code from Theorem 3.3.15 to obtain the following result.

Theorem 3.3.17. There exists an ≤ 2 -BSD permutation code of size at least $n!/n^3$.

The lower bound for the maximum size of an ≤ 2 -BSD permutation code from Theorem 3.3.17 is much larger compare to construction 3.3.9, which has size

$$\frac{2!n!}{((4 \cdot 2)!)^2 \cdot n^2} = \frac{1}{812851200} \times n!/n^3.$$

3.4 Permutation Codes Correcting a Burst of Stable Insertions

In this section, we design codes correcting a burst of insertions and discuss the equivalent relation between codes correcting stable deletions and codes correcting stable insertions. We first present the result on single insertion.

3.4.1 Optimal Codes Correcting a Single Insertion

Example 3.4.1. Suppose there is an insertion of symbol 2 occurring at the end in the permutation $(5, 3, 2, 1, 4)$, giving the vector $(5, 3, 2, 1, 4, 2)$. An error decoder need to identify the correct location of symbol 2, whether the third position or the last position. If we fix the locations of other symbols in a permutation π , and change the location of symbol 2, it is easy to see that any two of these following five permutations should not be in a code,

$$\{(2, 5, 3, 1, 4), (5, 2, 3, 1, 4), (5, 3, 2, 1, 4), (5, 3, 1, 2, 4), (5, 3, 1, 4, 2)\}.$$

For example, consider the first two permutations in the set. If there is an insertion of symbol 2 at the third position of the first permutation and an insertion of symbol 2 at

the beginning of the second permutation, we receive the same vector $(2, 5, 2, 3, 1, 4)$.

In general, for any $i \in [1, n]$, suppose $\pi = (\pi_1, \pi_2, \dots, \pi_{n-1}) \in \mathcal{S}_{n-1}$, where $\pi_j \in [1, n] \setminus \{i\}$ for $1 \leq j \leq n-1$, any two of the following n permutations in $\mathcal{S}_n$ should not be in a code $\{(i, \pi_1, \pi_2, \dots, \pi_{n-1}), (\pi_1, i, \pi_2, \dots, \pi_{n-1}), \dots, (\pi_1, \pi_2, \dots, \pi_{n-1}, i)\}$. We then have the following upper bound on size of an optimal code correcting a single insertion.

Theorem 3.4.2. Let $\mathcal{C}$ be a permutation codes capable of correcting a stable insertion. We have

$$|\mathcal{C}| \leq n!/n = (n-1)!$$

Theorem 3.4.3. Let $\mathcal{C}$ be a permutation codes $\mathcal{P}_a(n)$ for some $a \in \mathbb{Z}_n$, ($\mathcal{P}_a(n)$ was defined in subsection 3.2.1), which is capable of correcting a single stable deletion. Then $\mathcal{C}$ is also capable of correcting a single insertion.

Proof. We prove the correctness of the theorem by providing a method to recover a codeword if there is an insertion. Suppose there is an insertion occurred in σ , leads to a repetition of a symbol $i \in [1, n]$. To identify the correct location of symbol i , we first remove symbol i in both positions, and apply the decoding algorithm to recover σ from a vector of length $n-1$ containing all distinct symbols. It is doable since $\mathcal{C}$ is a permutation codes $\mathcal{P}_a(n)$ for some $a \in \mathbb{Z}_n$. There should be only one recovered codeword and the location for symbol i is then identified. $\square$

Since the union $\bigcup_a \mathcal{P}_a(n) = \mathcal{S}_n$, there exists a choice of a such that the size of $\mathcal{P}_a(n)$ is at least as average $n!/n = (n-1)!$. Combine to Theorem 3.4.2, this form an optimal code correcting a single insertion. We now discuss the duality between codes correcting a burst of stable deletions and codes correcting a burst of stable insertions. Specifically, we present a stronger result compared to Theorem 3.4.3 by showing the duality between these two codes when $s = 1$ and then generalize the result for $s \geq 2$.

Theorem 3.4.4. $\mathcal{C}$ is a permutation codes capable of correcting a stable deletion if and only if $\mathcal{C}$ is capable of correcting a stable insertion.

Proof. Let $\mathcal{D}_1(\pi)$ be the set of all vectors of length $n - 1$ received as a result of a stable deletion in π , and let $\mathcal{I}_1(\pi)$ be the set of all vectors of length $n + 1$ received as a result of a stable insertion in π . We now prove that for two permutations $\pi, \pi' \in \mathcal{S}_n$,

$$\mathcal{D}_1(\pi) \cap \mathcal{D}_1(\pi') = \emptyset \text{ iff } \mathcal{I}_1(\pi) \cap \mathcal{I}_1(\pi') = \emptyset$$

- Suppose that $\mathcal{D}_1(\pi) \cap \mathcal{D}_1(\pi') = \emptyset$, and we need to show $\mathcal{I}_1(\pi) \cap \mathcal{I}_1(\pi') = \emptyset$. We prove by contradiction. Suppose that $\mathcal{I}_1(\pi) \cap \mathcal{I}_1(\pi') \neq \emptyset$, in other words, there is a vector $\mathbf{x} \in \mathcal{I}_1(\pi) \cap \mathcal{I}_1(\pi')$, $\mathbf{x}$ is of length $n + 1$ and there is a repetition of symbol $i \in [1, n]$ at positions j_1, j_2 . Assume that $\mathbf{x} = (x_1, x_2, \dots, x_{j_1}, \dots, x_{j_2}, \dots, x_{n+1})$. Since $\mathbf{x} \in \mathcal{I}_1(\pi)$ and $\mathbf{x} \in \mathcal{I}_1(\pi')$, without loss of generality we have $\pi = (x_1, x_2, \dots, x_{j_1}, \dots, x_{j_2-1}, x_{j_2+1}, \dots, x_{n+1})$ and $\pi' = (x_1, x_2, \dots, x_{j_1-1}, x_{j_1+1}, \dots, x_{j_2}, \dots, x_{n+1})$. By deleting symbol i at position j_1 in π or at position j_2 in π' we receive $\mathbf{y} = (x_1, x_2, \dots, x_{j_1-1}, x_{j_1+1}, \dots, x_{j_2-1}, x_{j_2+1}, \dots, x_{n+1})$. Therefore, $\mathbf{y} \in \mathcal{D}_1(\pi) \cap \mathcal{D}_1(\pi')$. We have a contradiction since $\mathcal{D}_1(\pi) \cap \mathcal{D}_1(\pi') = \emptyset$.
- Similarly, suppose we have $\mathcal{I}_1(\pi) \cap \mathcal{I}_1(\pi') = \emptyset$, and we need to show $\mathcal{D}_1(\pi) \cap \mathcal{D}_1(\pi') = \emptyset$. We also prove by contradiction. Suppose $\mathcal{D}_1(\pi) \cap \mathcal{D}_1(\pi') \neq \emptyset$, in other words, there is a vector $\mathbf{x} \in \mathcal{D}_1(\pi) \cap \mathcal{D}_1(\pi')$ of length $n - 1$ and symbol $i \in [1, n]$ is deleted in π, π' . Since $\pi \neq \pi'$, the original positions of i in these two permutations are different. Assume that position of i in π is j_1 and in π' is j_2 . Since the order of other symbols are the same in these two permutations, leading an insertion of i in position j_2 in π would be the same as an insertion of i in position j_1 of π' . In other words, $\mathcal{I}_1(\pi) \cap \mathcal{I}_1(\pi') \neq \emptyset$, we have a contradiction.

Therefore $\mathcal{C}$ is a permutation codes capable of correcting a stable deletion if and only if $\mathcal{C}$ is capable of correcting a stable insertion. $\square$

3.4.2 s -BSD Codes and s -BSI Codes

Theorem 3.4.5. $\mathcal{C}$ is an s -BSD permutation code if and only if $\mathcal{C}$ is an s -BSI permutation code.

Proof. Let $\mathcal{D}_s(\pi)$ be the set of all vectors of length $n - s$ received as a result of a burst of s stable deletions in π , and let $\mathcal{I}_s(\pi)$ be the set of all vectors of length $n + s$ received as a result of a burst s stable insertions in π . We now prove that for two permutations $\pi, \pi' \in \mathcal{S}_n$,

$$\mathcal{D}_s(\pi) \cap \mathcal{D}_s(\pi') = \emptyset \text{ iff } \mathcal{I}_s(\pi) \cap \mathcal{I}_s(\pi') = \emptyset$$

- Suppose that $\mathcal{D}_s(\pi) \cap \mathcal{D}_s(\pi') = \emptyset$, we need to show $\mathcal{I}_s(\pi) \cap \mathcal{I}_s(\pi') = \emptyset$. We prove by contradiction. Suppose that $\mathcal{I}_s(\pi) \cap \mathcal{I}_s(\pi') \neq \emptyset$, in other words, there is a vector $\mathbf{x} \in \mathcal{I}_s(\pi) \cap \mathcal{I}_s(\pi')$, $\mathbf{x}$ is of length $n + s$ and there are repetitions of symbols $i_1, \dots, i_s \in [1, n]$. Assume that the positions of these symbols are $j_1, \dots, j_s$ (originally for $i_1, \dots, i_s$ in σ) and $k_1, k_1 + 1, \dots, k_1 + s$ for the inserted burst. On the other hand, assume that the positions of these symbols in π' are $j'_1, \dots, j'_s$ (originally in π') and $k'_1, k'_1 + 1, \dots, k'_1 + s$ for the inserted burst. Now, if $j_1, \dots, j_s$ are consecutive and $j'_1, \dots, j'_s$ are consecutive, then similar to the idea when $s = 1$, we can just remove all these symbols in two permutations of length $n + s$ to receive the same vector of length $n - s$, in other words, $\mathcal{D}_s(\pi) \cap \mathcal{D}_s(\pi') \neq \emptyset$, we then have a contradiction. Hence, $j_1, \dots, j_s$ are not consecutive and $j'_1, \dots, j'_s$ are not consecutive. Then from the resultant vector, we can identify where the errors occur, since there is only a burst inserted. By removing this burst of errors, we have $\pi \equiv \pi'$. We have a contradiction. Therefore, $\mathcal{I}_s(\pi) \cap \mathcal{I}_s(\pi') = \emptyset$.
- Similarly, suppose we have $\mathcal{I}_s(\pi) \cap \mathcal{I}_s(\pi') = \emptyset$, and we need to show $\mathcal{D}_s(\pi) \cap \mathcal{D}_s(\pi') = \emptyset$. This direction is easier to show the correctness. We also prove by

contradiction. Suppose $\mathcal{D}_s(\pi) \cap \mathcal{D}_s(\pi') \neq \emptyset$, in other words, there is a vector $\mathbf{x} \in \mathcal{D}_s(\pi) \cap \mathcal{D}_s(\pi')$ of length $n - s$ and symbol $i_1, i_2, \dots, i_s \in [1, n]$ deleted in π, π' . Since $\pi \neq \pi'$, the original locations of two deleted bursts in these two permutations are different. Assume that the burst in π occurs at positions $j_1, j_2, \dots, j_s$ and the burst in π' occurs at positions $j'_1, j'_2, \dots, j'_s$. Since the order of other symbols are the same in these two permutations, leading a burst of insertions in positions $j'_1, j'_2, \dots, j'_s$ in π would be the same as a burst of insertions in positions $j_1, j_2, \dots, j_s$ of π' . In other words, $\mathcal{I}_s(\pi) \cap \mathcal{I}_s(\pi') \neq \emptyset$, we have a contradiction.

Therefore $\mathcal{C}$ is an s -BSD permutation code if and only if $\mathcal{C}$ is an s -BSI permutation code. $\square$

3.4.3 s -BSI Codes

In previous subsection, we proved the equivalence of s -BSD codes and s -BSI codes. In other words, codes correcting a burst of stable deletions is also a code correcting a burst of stable insertions. Therefore, we also have the same results on upper bound and constructive lower bound for s -BSI codes.

Theorem 3.4.6. Let $n > s$ be positive integers. A code $\mathcal{C}$ is an s -BSI permutation code. We have

$$|\mathcal{C}| \leq \frac{n!}{s!(n-s+1)}.$$

The following results are implied from Corollary 3.3.8

Corollary 3.4.7. There exists an s -BSI permutation code of size at least $\frac{s}{((2s)!)^2} \cdot \frac{n!}{n^2}$.

Corollary 3.4.8. There exists a 2-BSI permutation code of size at least $\frac{n!}{n^2}$ and an ≤ 2 -BSI permutation code of size at least $\frac{n!}{n^3}$.

3.5 Permutation Codes Correcting a Burst of Unstable Deletions

Gabrys *et al.* proved that “for all $s \geq 1$, $\mathcal{C}$ is an s -BUD permutation code if and only if it is also an s -BUI permutation code. Therefore, code constructions for s -BUD permutation codes can be used for s -BUI permutation codes. An upper bound on permutation codes correcting an unstable deletion was presented by Gabrys *et al.* [19].

Lemma 3.5.1 (Gabrys et al. [19]). Let $\mathcal{C}$ be an 1-BUD permutation code. For any $0 < \epsilon < 1$, there exists an N_ϵ such that for all $n \geq N_\epsilon$, $|\mathcal{C}| \leq \frac{n!}{n(n-\log n)}(1 + \epsilon)$.

Moreover, they constructed codes of size at least $n!/n^2$, thus these codes are asymptotically optimal with respect to this upper bound. We now present code constructions by applying interleaving method to construct s -BUD permutation codes where $s \geq 2$.

Definition 3.5.2. For two vectors of length n , $\mathbf{a} = (a_1, a_2, \dots, a_n)$, $\mathbf{b} = (b_1, b_2, \dots, b_n)$, we define the *interleaved vector* $\mathbf{a} \circ \mathbf{b} \triangleq (a_1, b_1, a_2, b_2, \dots, a_n, b_n)$. In general, for m vectors of length n , $\mathbf{a}^i = (a_1^i, a_2^i, \dots, a_n^i)$ for $1 \leq i \leq m$, the *interleaved vector* $\mathbf{a}^1 \circ \mathbf{a}^2 \circ \dots \circ \mathbf{a}^m$ is obtained by alternatively placing the symbols of m vectors $\mathbf{a}^1, \mathbf{a}^2, \dots, \mathbf{a}^m$ in order.

Example 3.5.3. Let $m = 3, n = 3$, $\mathbf{a}^1 = (0, 1, 2)$, $\mathbf{a}^2 = (0, 3, 4)$, $\mathbf{a}^3 = (2, 1, 0)$. We have the interleave vector $\sigma = \mathbf{a}^1 \circ \mathbf{a}^2 \circ \mathbf{a}^3 = (0, 0, 2, 1, 3, 1, 2, 4, 0)$.

Definition 3.5.4. For a family of s codes $\mathcal{C}_i$, $i \in [1, s]$ of same length, the *interleaved code*

$$\mathcal{C}_1 \circ \mathcal{C}_2 \circ \dots \circ \mathcal{C}_s = \{\mathbf{c}^1 \circ \mathbf{c}^2 \circ \dots \circ \mathbf{c}^s : \mathbf{c}^i \in \mathcal{C}_i, i \in [1, s]\}.$$

Definition 3.5.5. For $k \in \mathbb{Z}$, a vector $\mathbf{a} = (a_1, a_2, \dots, a_n)$ and a code $\mathcal{C}$, we define the *translated vector* $\mathbf{a} + k = (a_1 + k, a_2 + k, \dots, a_n + k)$ and the *translated code* $\mathcal{C} + k = \{\mathbf{c} + k : \mathbf{c} \in \mathcal{C}\}$.

Recall that Gabrys *et al.* [19] constructed asymptotic optimal permutation codes that can correct a single unstable deletion as $\mathcal{P}_{a,b}(n)$ for given $a, b \in \mathbb{Z}_n$, defined as follows.

$$\mathcal{P}_{a,b}(n) = \left\{ \sigma \in S_n : \alpha(\sigma) \in \mathcal{C}_a(n-1) \text{ and } \alpha(\sigma^{-1}) \in \mathcal{C}_b(n-1) \right\}.$$

We now present our construction for s -BUD codes, where $s \geq 2$. We first construct 2-BUD permutation codes to explain the idea.

Theorem 3.5.6. Let $t \geq 3$ and $n = 2t$. Suppose that for each $i \in \{1, 2\}$, $\mathcal{C}_i \subseteq S_t$ is a permutation code which is capable of correcting a single unstable deletion. Then the interleaved code $\mathcal{C} = \mathcal{C}_1 \circ (\mathcal{C}_2 + t)$ is a 2-BUD code in S_n .

Proof. Suppose there are two adjacent unstable deletions in a codeword $\pi \in \mathcal{C}$, resulting $\tilde{\pi} \in \mathcal{S}_{n-2}$. We now provide an efficient error encoder to recover π . By the way we construct $\mathcal{C}$, $\pi = \mathbf{c}_1 \circ \mathbf{c}_2$ where $\mathbf{c}_1 \in \mathcal{C}_1, \mathbf{c}_2 \in \mathcal{C}_2$, and there is exactly one deletion in each of $\mathbf{c}_1$ and $\mathbf{c}_2$. To recover these two sequences, the decoder follows two steps.

Step 1: recover $\mathbf{c}_1$. Let $\mathbf{x} = (\tilde{\pi}_1, \tilde{\pi}_3, \dots, \tilde{\pi}_{n-3})$. We have $\mathbf{x} \in \mathcal{S}_{t-1}$, $\mathbf{x}$ is the resulting permutation when $\mathbf{c}_1$ experiences an unstable deletion. We can recover $\mathbf{c}_1$ from $\mathbf{x}$ since $\mathcal{C}_1$ is a permutation code capable of correcting a single unstable deletion.

Step 2: recover $\mathbf{c}_2$. Similarly, let $\mathbf{y} = (\tilde{\pi}_2, \tilde{\pi}_4, \dots, \tilde{\pi}_{n-2}) - (t-1)$. We have $\mathbf{y} \in \mathcal{S}_{t-1}$, $\mathbf{y}$ is the resulting permutation when $\mathbf{c}_2$ experiences an unstable deletion. We now recover $\mathbf{c}_2$ from $\mathbf{y}$. Finally, we recover $\pi = \mathbf{c}_1 \circ (\mathbf{c}_2 + t)$. $\square$

Surprisingly, we can use the code constructed in Theorem 3.5.7 to be an ≤ 2 -BUD permutation code, in other word, the code can correct up to 2 adjacent deletions.

Theorem 3.5.7. The code constructed in Theorem 3.5.7 is an ≤ 2 -BUD permutation code in S_n .

Proof. We have shown that the code is capable of correcting two adjacent deletions. We now modify our error decoder to handle the case when there is only one deletion.

Step 1: recover $\mathbf{c}_1$. Let $\mathbf{x}$ be the subsequence of $\tilde{\pi}$ with values from $[1, t-1]$. Notice that it is also a permutation in $\mathcal{S}_{t-1}$ and $\mathbf{x}$ is obtained from $\mathbf{c}_1$ by an unstable deletion, thus we can recover $\mathbf{c}_1$ from $\mathbf{x}$.

Step 2: recover $\mathbf{c}_2$. Similarly, Let $\mathbf{y}$ be the subsequence of $\tilde{\pi}$ with values from $[t+1, n-1]$. Let $\mathbf{z} = \mathbf{y} - t$. Thus $\mathbf{z}$ is the resulting permutation in $\mathcal{S}_{t-1}$ when $\mathbf{c}_2$ experiences an unstable deletion. We can recover $\mathbf{c}_2$ from $\mathbf{z}$.

Hence, we can recover π , which is the interleaved vector $\mathbf{c}_1 \circ (\mathbf{c}_2 + t)$ uniquely. $\square$

Example 3.5.8. Let $t = 5$ and $n = 10$. Suppose that $\mathbf{c}_1 = (5, 2, 1, 4, 3) \in \mathcal{C}_1$ and $\mathbf{c}_2 = (1, 3, 4, 5, 2) \in \mathcal{C}_2$. Then $\pi = \mathbf{c}_1 \circ (\mathbf{c}_2 + t) = (5, 6, 2, 8, 1, 9, 4, 10, 3, 7) \in \mathcal{C}$.

- If only one symbol 4 is deleted from π , then $\tilde{\pi} = (4, 5, 2, 7, 1, 8, 9, 3, 6)$ is received. We have $\mathbf{x} = (2, 1, 4, 3)$, and $\mathbf{y} = (6, 8, 9, 7)$. Then $\mathbf{z} = \mathbf{y} - 5 = (1, 3, 4, 2)$. we first recover $\mathbf{c}_1 = (5, 2, 1, 4, 3)$ and $\mathbf{c}_2 = (1, 3, 4, 5, 2)$ from $\mathbf{x}, \mathbf{z}$, respectively. Hence $(\mathbf{z} + t) = (6, 8, 9, 10, 7)$ and $\pi = \mathbf{c}_1 \circ (\mathbf{c}_2 + t) = (5, 6, 2, 8, 1, 9, 4, 10, 3, 7)$.
- If two symbols 10 and 4 are deleted, then $\tilde{\pi} = (4, 5, 2, 7, 1, 8, 3, 6)$ is received. We have $\mathbf{x} = (4, 2, 1, 3)$, and $\mathbf{y} = (5, 7, 8, 6) - 4 = (1, 3, 4, 2)$. From $\mathbf{x}$ and $\mathbf{y}$, again we can recover $\mathbf{c}_1 = (5, 2, 1, 4, 3)$ and $\mathbf{c}_2 = (1, 3, 4, 5, 2)$, respectively. Hence $\pi = \mathbf{c}_1 \circ (\mathbf{c}_2 + t)$ is also uniquely recovered.

The idea can be easily generalized to s -BUD permutation codes as follows.

Theorem 3.5.9. Let $t, s \geq 3$ and $n = st$. For each $i \in [1, s]$, let $\mathcal{C}_i$ be a permutation code over $[1, t]$ which is capable of correcting an unstable deletion. Then the interleaved code $\mathcal{C} = \mathcal{C}_1 \circ (\mathcal{C}_2 + t) \circ \cdots \circ (\mathcal{C}_i + (i-1)t) \circ \cdots \circ (\mathcal{C}_s + (s-1)t)$ is an s -BUD permutation code over $[1, n]$.

The code we construct in Theorem 3.5.9 has size at least $\left(\frac{(n/s)!}{(n/s)^2}\right)^s$ and its rate is asymptotically

$$\frac{s \ln(n/s)! - 2s \ln(n/s)}{\ln n!} \sim \frac{(n-2s) \ln n + O(n)}{n \ln n + O(n)} \sim 1, \text{ for fixed } s.$$

Chapter 4

Multipermutation and q -ary Codes

In this chapter, we present our results on *burst-deletion/insertion correcting codes* for *multipermutations*. The motivation of this work is the application of multipermutation codes in rank modulation scheme for flash memories as presented in Chapter 1 and Chapter 3. We then extend the techniques to construct *q -ary constant-composition* codes or generally *q -ary* codes that are capable of correcting a burst of deletions/insertions. We establish the correctness of our constructions by providing efficient error decoders that can recover original codewords from a burst of deletions or insertions.

4.1 Definitions and Notations

A multipermutation is a generalization of permutation where every element can appear multiple times.

Definition 4.1.1. For a positive integer n and a multiplicity (or composition) vector $\mathbf{r} = (r_1, r_2, \dots, r_m)$, where $n = \sum_{i=1}^m r_i$, we use $M(n, \mathbf{r})$ to denote the multiset

$$\{\underbrace{1, \dots, 1}_{r_1}, \underbrace{2, \dots, 2}_{r_2}, \dots, \underbrace{m, \dots, m}_{r_m}\}$$

A multipermutation is an arrangement of the elements of that multiset. We denote the set of all multipermutation on the multiset $M(n, \mathbf{r})$ as $\mathcal{S}_n^{\mathbf{r}}$. The size of $\mathcal{S}_n^{\mathbf{r}}$ is

$$\frac{n!}{r_1!r_2!\cdots r_m!}$$

If $r_1 = r_2 = \cdots = r_m = r$, the multiset and the corresponding multipermutations are called regular. For simplicity, throughout this chapter, we present our constructions for regular multipermutations. Our results for regular multipermutations can be easily generalized.

Example 4.1.2. Consider a multipermutation $\sigma = (3, 3, 2, 1, 2, 1, 2, 1)$ where the set of elements is $\{1, 2, 3\}$. The multiplicity vector is $\mathbf{r} = (3, 3, 2)$, where 3, 3, 2 are the number of times that symbol 1, 2 and 3 appear, respectively.

In multipermutations (or q -ary words), each symbol can appear multiple times, so we only consider deletion/insertion errors occur as stable case. In this chapter, *deletion/insertion* refer to stable deletion/insertion. For example, if there is a deletion at the third symbol of $(3, 3, \textcolor{red}{2}, 1, 2, 1, 2, 1)$, the resulting vector is $(3, 3, 1, 2, 1, 2, 1)$. We now define burst of deletions/insertions in multipermutation as follows.

Definition 4.1.3. Given $n \in \mathbb{Z}_+$ and a multiplicity vector $\mathbf{r} = (r_1, r_2, \dots, r_m)$, where $n = \sum_{i=1}^m r_i$, assume that $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_n) \in \mathcal{S}_n^{\mathbf{r}}$ and a set $I = \{i+1, \dots, i+s\} \subseteq [1, n]$, we say the multipermutation σ experiences a burst of s deletions in I , resulting in the vector $\tilde{\sigma} = (\sigma_1, \dots, \sigma_i, \sigma_{i+s+1}, \dots, \sigma_n)$.

For example, let $\sigma = (3, \textcolor{red}{3}, \textcolor{red}{1}, \textcolor{red}{2}, 2, 1, 2, 1) \in \mathcal{S}_8^{(3,3,2)}$ where $\mathbf{r} = (3, 3, 2)$ and $I = \{2, 3, 4\}$. If σ experiences a burst of s deletions in I , the resulting vector is $(3, 2, 1, 2, 1)$.

Definition 4.1.4. A code $\mathcal{C} \subseteq \mathcal{S}_n^{\mathbf{r}}$ is called an s -BD multipermutation code if it can correct a burst of exactly s deletions, or an $\leq s$ -BD multipermutation code if it can correct a burst of at most s deletions.

Similarly, we define s -BI and $\leq s$ -BI multipermutation codes as follows.

Definition 4.1.5. A code $\mathcal{C} \subseteq \mathcal{S}_n^r$ is called an s -BI multipermutation code if it can correct a burst of exactly s insertions, or an $\leq s$ -BI multipermutation code if it can correct a burst of at most s insertions.

Similar to a burst of deletions, if there is a burst of insertions in a multipermutation, from the resulting vector, we are able to tell the set of symbols that are inserted (or deleted). For example, if there is a burst of insertions occurred in $\sigma = (3, 3, 1, 2, 2, 1, 2, 1) \in \mathcal{S}_8^{(3,3,2)}$, resulting $\sigma' = (3, 3, 1, 2, 2, 2, 1, 3, 1, 2, 1)$, then the set of inserted symbols is $\{1, 2, 3\}$. Hence, we exploit this side information in our code constructions. The constructions for multipermutation codes can be applied to construct q -ary constant composition or q -ary balanced code capable of correcting a burst of deletions/insertions.

Recall that a multipermutation $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_n)$ of length $n = st$ for some $s, t \in \mathbb{Z}_+$ can also be written as an $s \times t$ array

$$\begin{bmatrix} \sigma_1 & \sigma_{s+1} & \cdots & \sigma_{(j-1)s+1} & \cdots & \sigma_{(t-1)s+1} \\ \sigma_2 & \sigma_{s+2} & \cdots & \sigma_{(j-1)s+2} & \cdots & \sigma_{(t-1)s+2} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \sigma_s & \sigma_{2s} & \cdots & \sigma_{js} & \cdots & \sigma_n \end{bmatrix}.$$

The i th row of an $s \times t$ multipermutation σ is denoted by

$$\sigma_{(s,i)} = (\sigma_i, \sigma_{s+i}, \dots, \sigma_{n-s+i}),$$

and the j th column of σ is denoted by

$$\sigma^{(s,j)} = (\sigma_{(j-1)s+1}, \sigma_{(j-1)s+2}, \dots, \sigma_{js}).$$

A burst of exactly s deletions/insertions deletes/inserts exactly one symbol in

each row of σ and the burst spans at most two adjacent columns of σ . Similar to permutation codes, we first prove the equivalence of *s-BD multipermutation codes* and *s-BI multipermutation codes*.

4.2 *s-BD* and *s-BI* multipermutation codes

Theorem 4.2.1. $\mathcal{C}$ is a multipermutation codes capable of correcting a burst of s deletions if and only if $\mathcal{C}$ is capable of correcting a burst of s insertions.

Proof. For a multipermutation $\sigma \in \mathcal{S}_n^r$, let $\mathcal{D}_s(\sigma)$ be the set of all vectors of length $n - s$ received when a burst of s deletions occur in σ , and let $\mathcal{I}_s(\sigma)$ be the set of all vectors of length $n + s$ received when a burst of s insertions occur in σ . We now prove that for two distinct multipermutations $\sigma, \sigma' \in \mathcal{S}_n^r$,

$$\mathcal{D}_s(\sigma) \cap \mathcal{D}_s(\sigma') = \emptyset \text{ if and only if } \mathcal{I}_s(\sigma) \cap \mathcal{I}_s(\sigma') = \emptyset$$

Suppose that we have $\mathcal{D}_s(\sigma) \cap \mathcal{D}_s(\sigma') = \emptyset$, and we need to show $\mathcal{I}_s(\sigma) \cap \mathcal{I}_s(\sigma') = \emptyset$. We prove by contradiction. Suppose that $\mathcal{I}_s(\sigma) \cap \mathcal{I}_s(\sigma') \neq \emptyset$, in other words, there is a vector $\mathbf{x} \in \mathcal{I}_s(\sigma) \cap \mathcal{I}_s(\sigma')$ of length $n + s$. Suppose the burst of insertions occurs from index $i + 1$ to $i + s$ in σ , and the other burst occurs from index $j + 1$ to $j + s$ in σ' .

1. If the two index ranges of two bursts do not intersect. In other words, $i + 1 > j + s$ or $i + s < j + 1$. Since the resultant vectors are the same, the original subsequence from index $i + 1$ to $i + s$ in σ' is the same as the inserted burst in σ and the original subsequence from index $j + 1$ to $j + s$ in σ is the same as the inserted burst in σ' . Removing this two subsequences in σ' and σ respectively we would have the same vector. Therefore, $\mathcal{D}_s(\sigma) \cap \mathcal{D}_s(\sigma') \neq \emptyset$, we have a contradiction. Thus, $\mathcal{I}_s(\sigma) \cap \mathcal{I}_s(\sigma') = \emptyset$.

2. If the two index ranges of two bursts intersect. Assume that $j + 1 \leq i + 1 \leq j + s \leq i + s$. It implies that the subsequence from index $j + 1$ to i in σ is the same as the subsequence from index $j + s + 1$ to $i + s$ in σ' . In other words, we can remove these two subsequences in σ' and σ respectively we would have the same vector. We may delete more symbols in the resulting vector to obtain a burst of exactly s deletions. In the end, we have $\mathcal{D}_s(\sigma) \cap \mathcal{D}_s(\sigma') \neq \emptyset$, which is a contradiction. Thus, $\mathcal{I}_s(\sigma) \cap \mathcal{I}_s(\sigma') = \emptyset$.

Similarly, suppose that we have $\mathcal{I}_s(\sigma) \cap \mathcal{I}_s(\sigma') = \emptyset$, and we need to show $\mathcal{D}_s(\sigma) \cap \mathcal{D}_s(\sigma') = \emptyset$. Suppose $\mathcal{D}_s(\sigma) \cap \mathcal{D}_s(\sigma') \neq \emptyset$, in other words, there is a vector $\mathbf{x} \in \mathcal{D}_s(\sigma) \cap \mathcal{D}_s(\sigma')$ of length $n - s$. Assume that position burst in σ is $j_1, \dots, j_1 + s$ and in σ' is $j'_1, \dots, j'_1 + s$. Since the order of other symbols are the same in these two multipermutations, leading an insertion of burst in positions $j'_1, \dots, j'_1 + s$ in σ would be the same as an insertion burst in position $j_1, \dots, j_1 + s$ of σ' . In other words, $\mathcal{I}_s(\sigma) \cap \mathcal{I}_s(\sigma') \neq \emptyset$, we have a contradiction. $\square$

Therefore a multipermutation code capable of correcting a burst of deletions can also correct a burst of insertions of same length. We first show a trivial result on asymptotically optimal multipermutation codes correcting a single deletion.

4.3 Multipermutation Codes Correcting a Single Deletion/Insertion

Theorem 4.3.1. Let $A(n, \mathbf{r})$ be the size of largest multipermutation code in $\mathcal{S}_n^{\mathbf{r}}$ that can correct a single deletion and insertion. Let $r_0 = \max_{1 \leq i \leq m} r_i$, we have

$$A(n, \mathbf{r}) \leq \frac{1}{n - r_0 + 1} \cdot \frac{n!}{r_1! r_2! \dots r_m!}$$

Construction 4.3.2. For a positive integer n , let $\mathbf{r} = (r_1, r_2, \dots, r_m)$ be a multiplicity vector, and $n = \sum_{i=1}^m r_i$. For $a \in \mathbb{Z}_n$, let

$$\mathcal{C}(n, \mathbf{r}, a) = \{\sigma \in \mathcal{S}_n^{\mathbf{r}} : \alpha(\sigma) \in C_a(n-1)\},$$

Then $\mathcal{C}(n, \mathbf{r}, a)$ is capable of correcting a single deletion or insertion. There exists a such that the code size is at least $\frac{1}{n} \times \frac{n!}{r_1!r_2!\dots r_m!}$.

Proof. Assume that a deletion occurs in a multipermutation $\sigma \in \mathcal{C}(n, \mathbf{r}, a)$. To recover σ , our error decoder runs in two steps.

Step 1: Identify the deleted symbol. This can be done simply by using the multiplicity vector $\mathbf{r}$.

Step 2: Recover σ . Since $\alpha(\sigma) \in C_a(n-1)$, the original location of the deleted symbol can be found efficiently by the works in [18, 30] (refer to Section 3.2.1).

Thus, we can recover the original multipermutation σ efficiently. On the other hand, since the union of all sets $\mathcal{C}(n, \mathbf{r}, a)$ is $\mathcal{S}_n^{\mathbf{r}}$, there exists a such that the size of constructed code is at least $\frac{1}{n} \cdot \frac{n!}{r_1!r_2!\dots r_m!}$. $\square$

This result is a trivial construction for multipermutation codes correcting a single deletion or insertion. We then have

$$\frac{1}{n - r_0 + 1} \cdot \frac{n!}{r_1!r_2!\dots r_m!} \geq A(n, \mathbf{r}) \geq \frac{1}{n} \cdot \frac{n!}{r_1!r_2!\dots r_m!},$$

If m is a fraction of n , in other words, each r_i is constant and very small compared to the length of the codeword n , then the family of constructed codes are asymptotically optimal.

4.4 Multipermutation Codes Correcting a Burst of Deletions/Insertions

4.4.1 On the Difficulty of Construction for Multipermutation Codes

To construct permutation codes correcting a burst of s deletions/insertions, we write any permutation of length n as an $s \times t$ array and use the fact that a burst of deletions/insertions spans at most two columns in this array. Therefore, we use the VT constraint (refer to Construction 3.3.6) for the first row of the array in order to find the location of the deleted symbol. Consequently, we can identify the two columns where a burst of deletions or insertions occurs. However, this technique does not work for multipermutations. In particular, the VT constraint in the first row will not imply the location of the deleted symbols, and therefore, we are unable to identify the two columns where the burst of deletions or insertions occurs. For example, if the first row of an $s \times t$ multipermutation is $(1, 2, 2, 2, 3, 3, 2, 2)$, and a symbol 2 is deleted, resulting the vector $(1, 2, 2, 3, 3, 2, 2)$, we are unable to identify the location of the deleted symbol 2, whether at 2nd, 3rd, or at 4th position, leading to a difficult problem to identify which two columns in the array that a burst of errors occurs. In fact, we are only able to locate the *run* of the deleted symbol (a run of symbol i in a sequence $\mathbf{x}$ is a subsequence of $\mathbf{x}$ whose symbols are identical to i). In the previous example, we can identify that the deleted symbol 2 was from the first run of 2, not the second run at the end of the original sequence. In many works in literature for q -ary codes correcting a burst of deletions/insertions ([31], [33]), the constraint for the first row is a *run-length-limited (RLL)* constraint. In other words, it limits the length of the runs of each symbols. In this work, we make use of the multiplicity vector $\mathbf{r}$, the *permutation projection* and the *permutation rank* functions as defined in Subsection 3.3.2 to overcome this difficulty. Recall that permutation projection and the permutation rank functions are previously defined only for vectors with different components. We now extend this definition to general q -ary words.

Definition 4.4.1. For an integer vector of length n , $\mathbf{u} = (u_1, u_2, \dots, u_n)$, the permutation projection of $\mathbf{u}$, denoted by $\mathbf{p}(\mathbf{u})$, is a permutation in $\mathcal{S}_n$ where:

$$\mathbf{p}(\mathbf{u})_i = |\{j : u_j < u_i, 1 \leq j \leq n\}| + |\{j : u_j = u_i, 1 \leq j \leq i\}|.$$

Consequently, the permutation rank function $\mu : \mathbb{Z}^n \rightarrow [1, n!]$ is defined by

$$\mu(\mathbf{u}) = \ell(\mathbf{p}(\mathbf{u})).$$

Example 4.4.2. Let $\mathbf{u} = (1, 1, 2, 2, 1, 2)$ and $\mathbf{v} = (1, 2, 1, 2, 1, 2)$. The permutation projections of $\mathbf{u}, \mathbf{v}$ are $\mathbf{p}(\mathbf{u}) = (1, 2, 4, 5, 3, 6)$, $\mathbf{p}(\mathbf{v}) = (1, 4, 2, 5, 3, 6)$.

Observe that we still can uniquely determine vector $\mathbf{u}$ when knowing the set of symbols in $\mathbf{u}$ and the value of $\mu(\mathbf{u})$.

Example 4.4.3. Let $\mu(\mathbf{u}) = 2$ and we know the set of symbols in $\mathbf{u}$ is $\{1, 1, 2, 2, 2, 3\}$. From $\mu(\mathbf{u}) = 2$ and $n = 6$, we have $\mathbf{p}(\mathbf{u}) = (1, 2, 3, 4, 6, 5)$. If the set of symbols in $\mathbf{u}$ is $\{1, 1, 2, 2, 2, 3\}$ then $\mathbf{u} = (1, 1, 2, 2, 3, 2)$.

4.4.2 Code Construction

Construction 4.4.4. Given $n = rm = (r+1)st$, t even. Let $a \in \mathbb{Z}_{(r+1)t}$, $b \in \mathbb{Z}_m$, and $c, d \in \mathbb{Z}_{(2s(r+1))!}$. Let $\mathcal{C}_s(n; r, a, b, c, d)$ be the set of all regular multipermutations $\sigma \in \mathcal{S}_n^r$ such that the following holds:

(i) When we view σ as an $s \times (r+1)t$ array, the first row of σ satisfies $\sigma_{(s,1)} \in \mathcal{C}_{a,b}(rt; m)$.

(ii) When we view σ as an $s(r+1) \times t$ array, the pairs of adjacent columns satisfy

$$(1) \sum_{i=1}^{t/2} \mu(\sigma^{(s(r+1), 2i-1)} || \sigma^{(s(r+1), 2i)}) \equiv c \pmod{(2s(r+1))!}, \text{ and}$$

$$(2) \sum_{i=1}^{t/2-1} \mu(\sigma^{(s(r+1), 2i)} || \sigma^{(s(r+1), 2i+1)}) \equiv d \pmod{(2s(r+1))!}.$$

Theorem 4.4.5. The multipermutation code $\mathcal{C}_s(n; r, a, b, c, d)$ from Construction 4.4.4 is a regular s -MBSD code over $\mathcal{S}_n^r$.

Proof. We prove the correctness of the theorem by providing a decoding algorithm for $\mathcal{C}_s(n; r, a, b, c, d)$ that recovers from a burst of exactly s stable deletions.

Suppose a burst of s stable deletions occurs in a codeword $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_n)$ in $\mathcal{C}_s(n; r, a, b, c, d)$, giving $\tilde{\sigma} = (\tilde{\sigma}_1, \tilde{\sigma}_2, \dots, \tilde{\sigma}_{n-s})$. Let $\rho = (\tilde{\sigma}_1, \tilde{\sigma}_{s+1}, \dots, \tilde{\sigma}_{((r+1)t-2)s+1})$. So ρ is the first row of the received word when we view it as an $s \times (r+1)t$ array. Since the first row of σ is a codeword of a single-deletion-correcting code by construction, we can recover the first row of σ from ρ . From the recovered row, we can also determine the run of the deleted symbol. The length of the run is at most r . We now view σ as an $s(r+1) \times t$ array and observe that the run of deleted symbols spans at most two adjacent columns of this array. We then use the second constraint to recover the burst of deletions as in the proof of Theorem 3.3.7. $\square$

Corollary 4.4.6. There exists a regular s -MBSD code of size at least

$$\frac{s}{((2s(r+1))!)^2} \cdot \frac{1}{n^2} \cdot \frac{n!}{(r!)^m}.$$

If m is a fraction of n , in other words, each r is constant and very small compared to the length of the codeword n , then the first term in the formula of the least size of constructed codes, $\frac{s}{((2s(r+1))!)^2}$ is a constant, and the rate of constructed codes are asymptotically optimal. Our constructed codes have size $\Theta(1/n^2)$ as much as $\mathcal{S}_n^r$. In other words, the redundancy of our constructed codes is $(2 \log n) + \Theta(1)$.

To construct codes of general multipermutations, i.e. where $\mathbf{r} = (r_1, r_2, \dots, r_m)$. We choose $r_{\max} = \max_{i \in [1, m]} r_i$ and follow Construction 4.4.4 by replacing r by $r_{\max}$. The following result is immediate.

Corollary 4.4.7. There exists an s -MBS code of size at least

$$\frac{s}{((2s(r_{\max} + 1))!)^2} \cdot \frac{1}{n^2} \cdot \frac{n!}{(r_1!)(r_2!) \cdots (r_m!)}.$$

Our code construction can be applied to construct some special classes of q -ary codes correcting a burst of deletions/insertions, such as q -ary constant composition codes, or specially q -ary balanced codes. These codes has several important applications in communications or storage systems that require non-binary alphabets such as coding for bandwidth-efficient channels, design of oligonucleotide sequences for DNA computing, and coding to mitigate the effect of charge leakage in flash memories ([2]). For example, in q -ary balanced codes, each symbol appears with same frequency as $h = n/q$, we then choose $r_0 = h = n/q$ and follow Construction 4.4.4.

4.5 Constructions for General q -ary Codes

In this section, we provide two constructions for q -ary codes of length n that are capable of correcting burst of s deletions/insertions where $s \geq 2$. When $n = st$, we also write a q -ary vector $\mathbf{x} = (x_1, x_2, \dots, x_n)$, where $x_i \in \Sigma_q$ as an $s \times t$ array,

$$\begin{bmatrix} x_1 & x_{s+1} & \cdots & x_{(j-1)s+1} & \cdots & x_{(t-1)s+1} \\ x_2 & x_{s+2} & \cdots & x_{(j-1)s+2} & \cdots & x_{(t-1)s+2} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ x_s & x_{2s} & \cdots & x_{js} & \cdots & x_n \end{bmatrix}.$$

The i th row is denoted by $\mathbf{x}_{(s,i)} = (x_i, x_{s+i}, \dots, x_{n-s+i})$, and the j th column is denoted by $\mathbf{x}^{(s,j)} = (x_{(j-1)s+1}, x_{(j-1)s+2}, \dots, x_{js})$. A burst deletes/inserts exactly one symbol in each row, and spans at most two adjacent columns. As mentioned earlier, the VT constraint on the first row does not imply the location of the deleted symbol. In fact, it only gives the information on the run of that symbol. We can overcome

this difficulty by making more constraints on the first row in such a way that we are able to identify the location of the deleted symbol. In binary codes, Cheng et al. [33] restricted the first row to be one of these words $(1, 0, 1, 0, \dots)$ or $(0, 1, 0, 1, \dots)$. In other words, they avoided a run of length one. In q -ary, we still can apply this idea for the first row, and the number of possible words are much more than two. In fact, there are exactly $q(q-1)^{n-1}$ possible words.

In the following constructions, we make use of the inversion function β defined in 3.3.12, and the permutation rank function μ defined in 3.3.5. For a vector $\mathbf{x} = (x_1, x_2, \dots, x_n)$, we define $\text{Sum}(\mathbf{x}) = x_1 + x_2 + \dots + x_n$.

Example 4.5.1. Let $\mathbf{x} = (1, 4, 10, 7)$, then $\beta(\mathbf{x}) = 1$ as there is only one inversion of the pair $(10, 7)$ that occurs and $\text{Sum}(\mathbf{x}) = 1 + 4 + 10 + 7 = 22$.

Construction 4.5.2. For a positive integer $n = ts$, let $a \in \mathbb{Z}_t$, $\mathbf{b} = (b_1, b_2, \dots, b_s)$ where $b_i \in \mathbb{Z}_q$ for $1 \leq i \leq s$, $c, d \in \mathbb{Z}_{(2s)!}$. Let $\mathcal{C}_s(n, a, \mathbf{b}, c, d; q)$ be the set of all q -ary words $\mathbf{x} = (x_1, x_2, \dots, x_n) \in \mathbb{Z}_q^n$ satisfies the following constraints:

- (i) The first row $\mathbf{x}_{(s,1)} = (x_1, x_{s+1}, \dots, x_{(t-1)s+1})$
 - (1) is a codeword of the nonbinary VT-code, that is $\mathbf{x}_{(s,1)} \in \mathcal{C}_{a,b_1}(t; q)$,
 - (2) satisfies that $x_{(j-1)s+1} \neq x_{js+1}$ for $j = 1, 2, \dots, t$,
- (ii) The other rows $\mathbf{x}_{(s,i)}$ where $2 \leq i \leq s$, satisfy $\text{Sum}(\mathbf{x}_{(s,i)}) \equiv b_i \pmod{q}$,
- (iii) The columns satisfy
 - (1) $\sum_{j=1}^{t/2} \mu(\mathbf{x}^{(s,2j-1)} || \mathbf{x}^{(s,2j)}) \equiv c \pmod{(2s)!}$, and
 - (2) $\sum_{j=1}^{t/2-1} \mu(\mathbf{x}^{(s,2j)} || \mathbf{x}^{(s,2j+1)}) \equiv d \pmod{(2s)!}$.

Theorem 4.5.3. The q -ary code $\mathcal{C}_s(n, a, \mathbf{b}, c, d; q)$ from Construction 4.5.2 can correct a burst of s deletions/insertions.

Proof. The first constraint helps us to recover the first row $\mathbf{x}_{(s,1)}$. Since the run of each symbol in the first row is at most one, we can identify the location of the deleted symbol. Next, from the second constraint on $(s-1)$ remaining rows, which is $\text{Sum}(\mathbf{x}_{(s,i)}) \equiv b_i \pmod{q}$, we are able to tell which symbol was deleted. Since the burst of deletions span at most two columns of $\mathbf{x}$, and these two columns are defined now based on the location of the deleted symbol in the first row. We then use the μ function to recover the burst (refer to Construction 3.3.6, 3.3.9). $\square$

Corollary 4.5.4. There exists an s -BD or s -BI q -ary code of size at least

$$\frac{sq(q-1)^{t-1}q^{n-t}}{nq^s(2s!)^2}$$

We now show that we can do better. Indeed, we can recover each deleted symbol in each row with less constraint.

Construction 4.5.5. For a positive integer $n = ts$, let $\mathbf{a} = (a_1, a_2, \dots, a_s)$, where $a_1 \in \mathbb{Z}_t, a_i \in \{0, 1\}$ for $2 \leq i \leq s$ and $\mathbf{b} = (b_1, b_2, \dots, b_s)$ where $b_i \in \mathbb{Z}_q$ for $1 \leq i \leq s$, let $\mathcal{C}_s(n, \mathbf{a}, \mathbf{b}; q)$ be the set of all q -ary words $\mathbf{x} = (x_1, x_2, \dots, x_n) \in \mathbb{Z}_q^n$ satisfies the following constraints:

- (i) The first row $\mathbf{x}_{(s,1)} = (x_1, x_{s+1}, \dots, x_{(t-1)s+1})$
 - (1) is a codeword of the nonbinary VT-code, that is $\mathbf{x}_{(s,1)} \in \mathcal{C}_{a_1, b_1}(t; q)$,
 - (2) satisfies that $x_{(j-1)s+1} \neq x_{js+1}$ for $j = 1, 2, \dots, t$,
- (ii) The other rows $\mathbf{x}_{(s,i)}$ where $2 \leq i \leq s$,
 - (1) $\text{Sum}(\mathbf{x}_{(s,i)}) \equiv b_i \pmod{q}$,
 - (2) $\beta(\mathbf{x}_{(s,i)}) \equiv a_i \pmod{2}$.

Theorem 4.5.6. The q -ary code $\mathcal{C}_s(n, \mathbf{a}, \mathbf{b}; q)$ from Construction 4.5.5 can correct a burst of deletions/insertions of length s .

Proof. We illustrate our main idea by providing a decoding algorithm. Suppose a codeword $\mathbf{x}$ experiences a burst of s deletions. The constraint on the first row, which is the same as Construction 4.5.2, helps us to recover the first row $\mathbf{x}_{(s,1)}$. Since the run in the first row is at most one, we can identify the location of the deleted symbol, which leads to only two possible positions for the deleted symbols in the remaining $(s - 1)$ rows. The constraint on the sum of symbols in each row helps us to identify which symbol was deleted while the other constraint, using β function, helps us to find its location (refer to the proof of Construction 3.3.14). $\square$

Corollary 4.5.7. There exists an s -BD or s -BI q -ary code of size at least

$$\frac{sq(q-1)^{t-1}q^{n-t}}{nq(2q)^{s-1}}.$$

Chapter 5

Codes Correcting Tandem Duplications

In this chapter, we present our results on q -ary codes correcting *tandem duplication* error, a special type of insertion, which is also known as *sticky insertion*. This work is motivated by applications that store data in living organisms [42–44] where codes correcting tandem duplications have been suggested as a solution to improve the reliability of data storage [45]. In addition, from the fact that tandem duplication is one of the most two common repeats that occur in more than 50% of human genome [39], we believe that studying tandem duplication may provide deeper understanding of how genetic information is stored and preserved in DNA throughout the course of nature.

5.1 Definitions and Notations

Let $\Sigma = \{0, 1, \dots, q-1\}$ be an alphabet of $q \geq 2$ symbols, and for a positive integer n , let Σ^n denote the set of all strings or words of length n over Σ , we now state the *tandem duplication* rules as follows.

For nonnegative integers $k \leq n$ and $i \leq n-k$, we define $T_{i,k} : \Sigma_q^n \rightarrow \Sigma_q^{n+k}$ such that

$$T_{i,k}(\mathbf{x}) = \mathbf{u}\mathbf{v}\mathbf{v}\mathbf{w}, \text{ where } \mathbf{x} = \mathbf{u}\mathbf{v}\mathbf{w}, |\mathbf{u}| = i, |\mathbf{v}| = k.$$

If a finite sequence of tandem duplications of length k is performed to obtain $\mathbf{y}$ from $\mathbf{x}$, then we say that $\mathbf{y}$ is a k -*descendant* of $\mathbf{x}$, or $\mathbf{x}$ is a k -*ancestor* of $\mathbf{y}$, and denote this relation by $\mathbf{x} \xrightarrow[k]{*} \mathbf{y}$. Formally, $\mathbf{x} \xrightarrow[k]{*} \mathbf{y}$ means that for some positive t , there exist t non-negative integers $i_1, i_2, \dots, i_t$ such that $\mathbf{y} = T_{i_t,k} \circ T_{i_{t-1},k} \circ \dots \circ T_{i_1,k}(\mathbf{x})$.

Given a sequence $\mathbf{x}$ and integer k , we consider the set of words that may be obtained from $\mathbf{x}$ via a finite number of tandem duplications of length k . We define the k -*descendant cone* of $\mathbf{x}$ to be the set of all k -descendants of $\mathbf{x}$ and denote this cone by $D_k^*(\mathbf{x})$. In other words,

$$D_k^*(\mathbf{x}) \triangleq \left\{ \mathbf{y} \in \Sigma_q^* \mid \mathbf{x} \xrightarrow[k]{*} \mathbf{y} \right\}.$$

We are interested in the case that tandem duplications whose lengths are upper bounded by an integer k and we extend the previous definitions. Formally, if a finite sequence of tandem duplications of length up to k is performed to obtain $\mathbf{y}$ from $\mathbf{x}$, then we say that $\mathbf{y}$ is a $\leq k$ -*descendant* of $\mathbf{x}$ or $\mathbf{x}$ is a $\leq k$ -*ancestor* of $\mathbf{y}$ and denote this relation by $\mathbf{x} \xrightarrow[\leq k]{*} \mathbf{y}$. Similarly we define $\leq k$ -*descendant cone* $D_{\leq k}^*(\mathbf{x})$ as follows.

$$D_{\leq k}^*(\mathbf{x}) \triangleq \left\{ \mathbf{y} \in \Sigma_q^* \mid \mathbf{x} \xrightarrow[\leq k]{*} \mathbf{y} \right\}.$$

Example 5.1.1. Consider $\mathbf{x} = 0121012$ over Σ_3 . Since $T_{1,3}(\mathbf{x}) = 0121\mathbf{121}012$ and $T_{0,2}(0121121012) = 01\mathbf{01}0121121012$, we write

$$0121012 \xrightarrow[3]{*} 0121121012 \text{ and } 0121012 \xrightarrow[\leq 3]{*} 010121121012.$$

Alternatively, we have that $0121121012 \in D_3^*(\mathbf{x})$, and $010121121012 \in D_{\leq 3}^*(\mathbf{x})$.

Motivated by applications that store data on living organisms, Jain et al. [45] looked at the $\leq k$ -descendant cones of a pair of words and tried to determine whether the two cones have a nontrivial intersection. We first introduce the notion of confusability and tandem duplication codes.

5.2 Unbounded Tandem Duplication Codes

We aim to design codes that are capable of correcting tandem duplications of length up to a fixed value k , namely $\leq k$ -*tandem duplication codes*. We also call these codes as $\leq k$ -*unbounded tandem duplication codes*, meaning that there is no limit on the number of duplications that occur. In other words, there is no limit in the length of words in descendant cone of each word.

Definition 5.2.1 (Confusability). Two words $\mathbf{x}$ and $\mathbf{y}$, are said to be $\leq k$ -*confusable* if $D_{\leq k}^*(\mathbf{x}) \cap D_{\leq k}^*(\mathbf{y}) \neq \emptyset$.

To design error-correcting codes, we should use those codewords that are not pairwise confusable. Formally, we have the following definition of $\leq k$ -unbounded tandem duplication codes.

Definition 5.2.2 ($\leq k$ -Tandem-Duplication Codes). A subset $\mathcal{C} \subseteq \Sigma_q^n$ is a $\leq k$ -*unbounded tandem duplication code* or shortly $\leq k$ -*tandem duplication code* if for all $\mathbf{x}, \mathbf{y} \in \mathcal{C}$ and $\mathbf{x} \neq \mathbf{y}$, we have that $\mathbf{x}$ and $\mathbf{y}$ are not $\leq k$ -confusable. We say that $\mathcal{C}$ is an $(n, \leq k; q)$ -tandem duplication code or $(n, \leq k; q)$ -TD code.

To determine if a set of words is a tandem duplication code, we need to verify that all pairs of distinct words are not confusable. Surprisingly, the answer to this problem is only known for $k \leq 2$ and there is no result for larger k .

5.3 Previous Results and Open Problems

5.3.1 Main Idea

Confusability. We summarize known efficient algorithms that determine whether two words are $\leq k$ -confusable for $k \in \{1, 2\}$ and we then explain why these methods cannot be extended for the case $k \geq 3$. Crucial to the algorithms is the concept of irreducible words and roots.

Definition 5.3.1. A word $\mathbf{x}$ is said to be $\leq k$ -irreducible if $\mathbf{x}$ cannot be deduplicated into shorter words with deduplication of length at most k . In other words, if $\mathbf{y} \xrightarrow[\leq k]{*} \mathbf{x}$, then $\mathbf{y} = \mathbf{x}$. The set of $\leq k$ -irreducible q -ary words is denoted by $\text{Irr}_{\leq k}(q)$ and the set of $\leq k$ -irreducible words with length n is denoted by $\text{Irr}_{\leq k}(n, q)$. The $\leq k$ -ancestors of $\mathbf{x} \in \Sigma_q^*$ that are $\leq k$ -irreducible words are called the $\leq k$ -roots of $\mathbf{x}$. The set of $\leq k$ -roots of $\mathbf{x}$ is denoted by $R_{\leq k}(\mathbf{x})$.

Example 5.3.2. For alphabet Σ_3 , we have $R_{\leq 3}(0101222) = \{012\}$, and $R_{\leq 2}(012012) = \{012012\}$. On the other hand, $R_{\leq 4}(012101212) = \{012, 0121012\}$ as

$$\begin{aligned} 012 &\xrightarrow[2]{*} 012\textcolor{red}{1}2 \xrightarrow[4]{*} 0121\textcolor{red}{0}1\textcolor{red}{2}12 \text{ and} \\ 0121012 &\xrightarrow[2]{*} 0121012\textcolor{red}{1}2. \end{aligned}$$

From the above example, it is possible for a word to have more than one root. Jain *et al.* [45] determined when a word has exactly one root.

Proposition 5.3.3 (Jain et al. [45]). For any $\mathbf{x} \in \Sigma_q^*$, we have that $|R_{\leq k}(\mathbf{x})| = 1$ for all $k \in \{1, 2, 3\}$.

Surprisingly, for certain cases, determining the confusability of two words is equivalent to computing the roots for the words. Specifically, two words are $\leq k$ -confusable where $k \leq 2$ if and only if they have the same root.

Proposition 5.3.4 (Jain et al. [45]). For all $\mathbf{x}, \mathbf{y} \in \Sigma_q^*$, we have that

- (i) $\mathbf{x}$ and $\mathbf{y}$ are $\leq k$ -confusable if and only if $R_{\leq k}(\mathbf{x}) = R_{\leq k}(\mathbf{y})$ for $k \in \{1, 2\}$.
- (ii) If $R_{\leq 3}(\mathbf{x}) \neq R_{\leq 3}(\mathbf{y})$, then $\mathbf{x}$ and $\mathbf{y}$ are not ≤ 3 -confusable.

In other words, when $k \in \{1, 2\}$, to determine $\mathbf{x}$ and $\mathbf{y}$ are $\leq k$ -confusable, it is both necessary and sufficient to compute the $\leq k$ -roots of $\mathbf{x}$ and $\mathbf{y}$. One may derive a linear-time algorithm to compute the $\leq k$ -roots of any word $\mathbf{x}$, and therefore, applying Proposition 5.3.3 we can obtain linear-time algorithms that determine whether two words are $\leq k$ -confusable for all $k \in \{1, 2\}$.

Unfortunately, when $k = 3$, it is no longer sufficient to compute the ≤ 3 -roots of $\mathbf{x}$ and $\mathbf{y}$. Specifically, even though $R_{\leq 3}(\mathbf{x}) = R_{\leq 3}(\mathbf{y})$, it is still possible that $\mathbf{x}$ and $\mathbf{y}$ are not ≤ 3 -confusable. A counterexample is as follows.

Example 5.3.5. Consider $\mathbf{x} = 012012$ and $\mathbf{y} = 011112$ over Σ_3 . The words have the same root as $R_{\leq 3}(\mathbf{x}) = R_{\leq 3}(\mathbf{y}) = \{012\}$. However, $\mathbf{x}$ and $\mathbf{y}$ are not ≤ 3 -confusable because any ≤ 3 -descendant of $\mathbf{x}$ has a 2 to the left of a 0, whereas any ≤ 3 -descendant of $\mathbf{y}$ does not.

Therefore, the next smallest open case is where $k = 3$. Even though the first statement in Proposition 5.3.4 fails when $k = 3$, the second statement suggests a method to construct tandem duplication codes by using all words with different roots. For example, to construct a code of length n , one may easily collect all words of length n that have different roots. Since each root is an irreducible word of length from one to n , the code's size is actually the size of union of all irreducible words of length from one to n . The natural question is how good these codes are. In this chapter, we

also study the set of irreducible words and surprisingly, the set of irreducible words of length n can form a good $(n, \leq k; q)$ -TD code for $k \in \{1, 2, 3\}$. We first summarize the technique to construct $(n, \leq k; q)$ -TD code for $k \in \{1, 2, 3\}$ in previous works.

5.3.2 Previous Constructions

Using Proposition 5.3.4, Jain et al. [45] used irreducible words to construct tandem duplication codes.

Construction 5.3.6. For $k \in \{2, 3\}$ and $n \geq k$. An $(n, \leq k; q)$ -TD code $\mathcal{C}_{\text{Irr}}(n, \leq k; q)$ is given by

$$\mathcal{C}_{\text{Irr}}(n, \leq k; 3) = \bigcup_{i=1}^n \{\xi_{n-i}(\mathbf{x}) \mid \mathbf{x} \in \text{Irr}_{\leq k}(i, q)\},$$

where $\xi_i(\mathbf{x}) = \mathbf{x}z^i$, and z is the last symbol of $\mathbf{x}$. In other words, $\xi_i(\mathbf{x})$ is the sequence $\mathbf{x}$ with its last symbol z repeated i more times. The repetition ensures that all codewords in $\mathcal{C}_{\text{Irr}}(n, \leq k; 3)$ are of equal length n . Moreover, the size of $\mathcal{C}_{\text{Irr}}(n, \leq k; q)$ is $\sum_{i=1}^n |\text{Irr}_{\leq k}(i, q)|$.

It then follows from the first statement in Proposition 5.3.4 that $\mathcal{C}_{\text{Irr}}(n, \leq 2; q)$ is an optimal $(n, \leq 2; q)$ -TD code. Unfortunately, the TD code $\mathcal{C}_{\text{Irr}}(n, \leq 3; q)$ is not optimal for $n \geq 6$.

Example 5.3.7. Consider $n = 6$, $q = 3$. The code $\mathcal{C}_{\text{Irr}}(6, \leq 3; 3)$ from Construction 5.3.6 is given by

$$\begin{aligned} \mathcal{C}_{\text{Irr}}(6, \leq 3; q) &= \{aaaaaa\} \cup \{abbbbb\} \\ &\cup \{abaaaa, abcccc\} \\ &\cup \{abaccc, abcaaa, abcbbb\} \\ &\cup \{abacaa, abacbb, abcabb, abcacc, abcbaa\} \\ &\cup \{abacab, abacba, abacbc, abcaba, abcacb, abcbab, abcbac\}, \end{aligned}$$

where a, b, c refer to a template, and can be substituted with $0, 1, 2$ in any permutation. From this construction, $|\mathcal{C}_{\text{irr}}(6, \leq 3; 3)| = 111$. However, if we remove from $\mathcal{C}_{\text{irr}}(6, \leq 3; 3)$ the six codewords of the form $\{abcccc\}$ and add to the code with twelve more codewords of the forms $\{abcabc, abbbbc\}$, then the new code has size 117.

5.3.3 Open Problems and Our Main Results

Codes constructed from Construction 5.3.6 is optimal when $k = 1, 2$. When $k = 3$, even though the TD code $\mathcal{C}_{\text{irr}}(n, \leq 3; q)$ is not optimal for $n \geq 6$, we later show that its rate is very closed to the optimal rate, especially the gap between its rate and the optimal rate is smaller than 0.01 for $q \geq 5$. There is no known construction for codes correcting tandem duplication of length more than or equal to four ($k \geq 4$). Our main results on this chapter focus on:

- **Efficient encoding/decoding of irreducible words:** all known codes constructions are based on irreducible words, and optimal codes are obtained when $k \in \{1, 2\}$. When $k = 3$, codes constructed using Construction 5.3.6 is not optimal. In Section 5.4 we provide a detail analysis to show that such codes not only achieve good rate but they also have efficient linear decoding algorithms. Consequently, in Section 5.5 we provide two efficient encoding/decoding algorithms for such codes, using finite state encoder and enumerative coding. This marks the first known efficient encoder/decoder for codes correcting tandem duplications.
- **Constructions for $(n, \leq k; q)$ -TD codes when $k \geq 4$:** There is no known construction for codes correcting tandem duplication of length more than or equal to four ($k \geq 4$). Even though the set of all irreducible words does not form a code, we strongly believe that irreducible words still play some important role to design TD codes. Throughout Section 5.6, we present a construction for

$(n, \leq k; q)$ -TD codes where $k \geq 4$. This is not just the first known construction when $k \geq 4$, the code is a subset of irreducible words, and moreover, its rate is close to the upper bound. We establish the correctness of our construction by providing an efficient error decoder that runs in time linear in the length of codewords.

5.4 Irreducible Words and Our Conjecture

In this section, we provide a detailed analysis on $\mathcal{C}_{\text{Irr}}(n, \leq k; q)$ where $k \in \{2, 3\}$. Recall that the set of $\leq k$ -irreducible words with length n is denoted by $\text{Irr}_{\leq k}(n, q)$. Let $I_{\leq k}(n, q) \triangleq |\text{Irr}_{\leq k}(n, q)|$. Then the size of $\mathcal{C}_{\text{Irr}}(n, \leq k; q)$ is given by $\sum_{i=1}^n I_{\leq k}(i, q)$. Let $\text{rate}_{\leq k}(n, q)$ and $\text{rate}_{\leq k}(q)$ denote the rate and asymptotic rate of $\mathcal{C}_{\text{Irr}}(n, \leq k; q)$, respectively. In other words,

$$\begin{aligned} \text{rate}_{\leq k}(n, q) &\triangleq (1/n) \log_q |\mathcal{C}_{\text{Irr}}(n, \leq k; q)|, \text{ and} \\ \text{rate}_{\leq k}(q) &\triangleq \lim_{n \rightarrow \infty} \text{rate}_{\leq k}(n, q). \end{aligned}$$

Jain *et al.* [45] show that $\bigcup_{n=1}^{\infty} \text{Irr}_{\leq k}(n, q)$ is a regular language and therefore,

$$\text{rate}_{\leq k}(q) = \lim_{n \rightarrow \infty} \frac{\log_q I_{\leq k}(n, q)}{n}.$$

We first introduce a recursive formula for $I_{\leq k}(n, q)$ where $k \in \{2, 3\}$ and consequently, provide a formula for the asymptotic rate for $\mathcal{C}_{\text{Irr}}(n, \leq k; q)$. Our main idea is to partition the set of irreducible words into two classes and provide bijections from irreducible words of shorter lengths into them. In other words, we aim to compute $I_{\leq k}(n, q)$ from certain $I_{\leq k}(m, q)$ where $m < n$.

Specifically, notice that the suffix of an irreducible word is of the form either aba or abc , where a, b, c are distinct symbols. We then let $\text{Irr}_{\leq k}^{(s)}(2, n, q)$ and $\text{Irr}_{\leq k}^{(s)}(3, n, q)$ denote the set of irreducible words with length-three suffixes that have two and three dis-

tinct symbols, respectively. The size of these sets are denoted by $I_{\leq k}^{(s)}(2, n, q)$, $I_{\leq k}^{(s)}(3, n, q)$, respectively. We then have

$$I_{\leq k}(n, q) = I_{\leq k}^{(s)}(2, n, q) + I_{\leq k}^{(s)}(3, n, q).$$

5.4.1 Results on $k = 2$

When $k = 2$, we consider the following maps for $n \geq 4$,

$$\begin{aligned}\phi &: \text{Irr}_{\leq 2}(n-1, q) \times [q-2] \rightarrow \text{Irr}_{\leq 2}^{(s)}(3, n, q), \\ \psi &: \text{Irr}_{\leq 2}(n-2, q) \times [q-2] \rightarrow \text{Irr}_{\leq 2}^{(s)}(2, n, q).\end{aligned}$$

We first define ϕ . If $\mathbf{x} = x_1 x_2 \dots x_{n-1} \in \text{Irr}_{\leq 2}(n-1, q)$ and $i \in [q-2]$, set σ to be the i th element in $\Sigma_q \setminus \{x_{n-2}, x_{n-1}\}$. Then set $\phi(\mathbf{x}, i) = x_1 x_2 \dots x_{n-1} \sigma$.

For ψ , let $\mathbf{x} = x_1 x_2 \dots x_{n-2} \in \text{Irr}_{\leq 2}(n-2, q)$ and $i \in [q-2]$ and set σ to be the i th element in $\Sigma_q \setminus \{x_{n-3}, x_{n-2}\}$. Then set $\psi(\mathbf{x}, i) = x_1 x_2 \dots x_{n-2} \sigma x_{n-2}$.

Proposition 5.4.1. The maps ϕ and ψ are bijections.

Proof. We construct the inverse map for ϕ . Specifically, we set $\phi^{-1} : \text{Irr}_{\leq 2}^{(s)}(3, n, q) \rightarrow \text{Irr}_{\leq 2}(n-1, q) \times [q-2]$ such that $\phi^{-1}(\mathbf{x}) = (x_1 \dots x_{n-1}, i)$, where i is the index of x_n in $\Sigma_q \setminus \{x_{n-2}, x_{n-1}\}$. It can be verified that $\phi \circ \phi^{-1}$ and $\phi^{-1} \circ \phi$ are identity maps on their respective domains. Therefore, ϕ is a bijection map and similarly, we can also prove that ψ is bijective. $\square$

Corollary 5.4.2. We have that $I_{\leq 2}(2, q) = q(q-1)$, $I_{\leq 2}(3, q) = q(q-1)^2$, and

$$I_{\leq 2}(n, q) = (q-2)I_{\leq 2}(n-1, q) + (q-2)I_{\leq 2}(n-2, q) \quad (5.1)$$

for $n \geq 4$. Therefore, the asymptotic rate is $\text{rate}_{\leq 2}(q) = \log_q \lambda_2$, where $\lambda_2 = (q-2 + \sqrt{q^2-4})/2$.

Fix $\mathbf{p}$. We observe that the maps ϕ and ψ only append one and two symbols to words in their domains, respectively. So, if we apply the maps to a word with prefix $\mathbf{p}$, the image also has the same prefix $\mathbf{p}$. Consequently, ϕ and ψ still remain as bijections when we restrict the domains and codomains to the irreducible words with prefix $\mathbf{p}$.

Corollary 5.4.3. Let $\mathbf{p} \in \Sigma_q^\ell$ For $n \geq \ell + 2$,

$$|\text{Irr}_{\leq 2}^{(p)}(\mathbf{p}, n, q)| = (q - 2) |\text{Irr}_{\leq 2}^{(p)}(\mathbf{p}, n - 1, q)| + (q - 2) |\text{Irr}_{\leq 2}^{(p)}(\mathbf{p}, n - 2, q)|. \quad (5.2)$$

5.4.2 Results on $k = 3$

We now present the recurrence for $\text{Irr}_{\leq 3}(n, q)$.

Proposition 5.4.4. We have that $I_{\leq 3}(3, q) = q(q - 1)^2$, $I_{\leq 3}(4, q) = q^2(q - 1)(q - 2)$, $I_{\leq 3}(5, q) = q(q - 1)(q - 2)(q^2 - q - 1)$, and for $n \geq 6$,

$$I_{\leq 3}(n, q) = (q - 2)I_{\leq 3}(n - 1, q) + (q - 3)I_{\leq 3}(n - 2, q) + (q - 2)I_{\leq 3}(n - 3, q) \quad (5.3)$$

Consequently, $\text{rate}_{\leq 3}(q) = \log_q \lambda_3$, where λ_3 is the largest real root of equation $x^3 - (q - 2)x^2 - (q - 3)x - (q - 2) = 0$.

Proof. For a word $\mathbf{p}$ of length $\ell < n$, let $\text{Irr}_{\leq k}^{(s)}(\mathbf{p}, n, q)$ be the set of irreducible words of length n with suffix $\mathbf{p}$. Let $\mathbf{L}$ be a set of certain words $\mathbf{p}$, we denote $\text{Irr}_{\leq k}^{(s)}(\mathbf{L}, n, q)$ is the set of irreducible words of length n with suffix in $\mathbf{L}$. Similar to the case when $k = 2$, to prove this proposition, we partition the set of irreducible words $\text{Irr}_{\leq 3}(n, q)$ into three classes and provide bijections from irreducible words of shorter length into them. Specifically, we consider all possible suffixes of length six of an irreducible word. For a word $\mathbf{x} = x_1 \dots x_{n-5}x_{n-4}x_{n-3}x_{n-2}x_{n-1}x_n \in \text{Irr}_{\leq 3}(n, q)$, if $x_n = x_{n-3} = a$, every suffix of length six must be of the form $\{bcabda, bcacba, bcacda, abacba, abacda\}$, where a, b, c, d are different elements in Σ_q . On the other hand, if $x_n \neq x_{n-3}$, every suffix of length four must be of the form $\{abcd, abcb, abac\}$. Let $\mathbf{L}_1 = \{abcd, abcb, abac | a, b, c, d \in \Sigma_q\}$.

$\Sigma_q\}$, $\mathbf{L}_2 = \{bcabda, bcacba, bcacda | a, b, c, d \in \Sigma_q\}$, $\mathbf{L}_3 = \{abacba, abacda | a, b, c, d \in \Sigma_q\}$. We consider the following maps for $n \geq 6$.

$$\begin{aligned}\varphi_1 &: \text{Irr}_{\leq 3}(n-1, q) \times [q-2] \rightarrow \text{Irr}_{\leq 3}^{(s)}(\mathbf{L}_1, n, q), \\ \varphi_2 &: \text{Irr}_{\leq 3}(n-2, q) \times [q-2] \rightarrow \text{Irr}_{\leq 3}^{(s)}(\mathbf{L}_2, n, q), \\ \varphi_3 &: \text{Irr}_{\leq 3}(n-3, q) \times [q-3] \rightarrow \text{Irr}_{\leq 3}^{(s)}(\mathbf{L}_3, n, q).\end{aligned}$$

Recall that $\text{Irr}_{\leq k}(n, q) = \text{Irr}_{\leq k}^{(s)}(3, n, q) \cup \text{Irr}_{\leq k}^{(s)}(2, n, q)$.

If $\mathbf{x} = x_1 \dots x_{n-3}x_{n-2}x_{n-1} \in \text{Irr}_{\leq 3}^{(s)}(3, n-1, q)$ and $i \in [q-2]$, set σ to be the i th element in $\Sigma_q \setminus \{x_{n-3}, x_{n-1}\}$. Then set

$$\varphi_1(\mathbf{x}, i) = \mathbf{x}\sigma = x_1 \dots x_{n-3}x_{n-2}x_{n-1}\sigma.$$

If $\mathbf{x} = x_1 \dots x_{n-3}x_{n-2}x_{n-1} \in \text{Irr}_{\leq 3}^{(s)}(2, n-1, q)$ where $x_{n-3} = x_{n-1}$ and $i \in [q-2]$, set σ to be the i th element in $\Sigma_q \setminus \{x_{n-2}, x_{n-1}\}$.

$$\varphi_1(\mathbf{x}, i) = \mathbf{x}\sigma = x_1 \dots x_{n-1}x_{n-2}x_{n-1}\sigma.$$

If $\mathbf{x} = x_1 \dots x_{n-5}x_{n-4}x_{n-3}x_{n-2} \in \text{Irr}_{\leq 3}^{(s)}(3, n-2, q)$ and $i \in [q-3]$, we set

$$\varphi_2(\mathbf{x}, i) = \mathbf{x}\sigma x_{n-3} = x_1 \dots x_{n-5}x_{n-4}x_{n-3}x_{n-2}\sigma x_{n-3},$$

where σ is the i th element in $\Sigma_q \setminus \{x_{n-5}, x_{n-3}, x_{n-2}\}$ if $x_{n-5} \notin \{x_{n-2}, x_{n-3}\}$ or the i th element in $\Sigma_q \setminus \{x_{n-4}, x_{n-3}, x_{n-2}\}$ if $x_{n-5} \in \{x_{n-2}, x_{n-3}\}$.

If $\mathbf{x} = x_1 \dots x_{n-5}x_{n-4}x_{n-3}x_{n-2} \in \text{Irr}_{\leq 3}^{(s)}(2, n-2, q)$ where $x_{n-4} = x_{n-2}$ and $i \in [q-3]$, set σ to be the i th element in $\Sigma_q \setminus \{x_{n-5}, x_{n-3}, x_{n-2}\}$ and

$$\varphi_2(\mathbf{x}, i) = \mathbf{x}\sigma x_{n-3} = x_1 \dots x_{n-5}x_{n-4}x_{n-3}x_{n-2}\sigma x_{n-3}.$$

If $\mathbf{x} = x_1 \dots x_{n-5}x_{n-4}x_{n-3} \in \text{Irr}_{\leq 3}^{(s)}(3, n-3, q)$ and $i \in [q-2]$, set σ to be the i th element in $\Sigma_q \setminus \{x_{n-5}, x_{n-3}\}$. Then set

$$\varphi_3(\mathbf{x}, i) = \mathbf{x}\sigma x_{n-5}x_{n-3} = x_1 \dots x_{n-5}x_{n-4}x_{n-3}\sigma x_{n-5}x_{n-3}.$$

If $\mathbf{x} = x_1 \dots x_{n-5}x_{n-4}x_{n-3} \in \text{Irr}_{\leq 3}^{(s)}(2, n-3, q)$ where $x_{n-5} = x_{n-3}$ and $i \in [q-2]$, set σ to be the i th element in $\Sigma_q \setminus \{x_{n-4}, x_{n-3}\}$. Then set

$$\varphi_3(\mathbf{x}, i) = \mathbf{x}\sigma x_{n-4}x_{n-3} = x_1 \dots x_{n-5}x_{n-4}x_{n-3}\sigma x_{n-4}x_{n-3}.$$

We can prove that φ_i is bijection for $i = 1, 2, 3$ by constructing the inverse maps.

$$\begin{aligned}\varphi_1^{-1} : \text{Irr}_{\leq 3}^{(s)}(\mathbf{L}_1, n, q) &\rightarrow \text{Irr}_{\leq 3}(n-1, q) \times [q-2] \\ \varphi_2^{-1} : \text{Irr}_{\leq 3}^{(s)}(\mathbf{L}_2, n, q) &\rightarrow \text{Irr}_{\leq 3}(n-2, q) \times [q-2] \\ \varphi_3^{-1} : \text{Irr}_{\leq 3}^{(s)}(\mathbf{L}_3, n, q) &\rightarrow \text{Irr}_{\leq 3}(n-3, q) \times [q-3],\end{aligned}$$

where $\varphi_1^{-1}(\mathbf{x}) = (x_1 \dots x_{n-3}x_{n-2}x_{n-1}, i)$ where i is the index of x_n in

- $\Sigma_q \setminus \{x_{n-3}, x_{n-1}\}$ if $x_{n-1} \neq x_{n-3}$, or
- $\Sigma_q \setminus \{x_{n-2}, x_{n-1}\}$ otherwise,

and $\varphi_2^{-1}(\mathbf{x}) = (x_1 \dots x_{n-3}x_{n-2}, i)$ where i is the index of x_{n-1} in

- $\Sigma_q \setminus \{x_{n-5}, x_{n-3}, x_{n-2}\}$ if $x_{n-5} \notin \{x_{n-3}, x_{n-2}\}$ or $x_{n-4} = x_{n-2}$, or
- $\Sigma_q \setminus \{x_{n-4}, x_{n-3}, x_{n-2}\}$ if $x_{n-5} \in \{x_{n-3}, x_{n-2}\}$ and $x_{n-4} \neq x_{n-2}$,

and lastly, $\varphi_3^{-1}(\mathbf{x}) = (x_1 \dots x_{n-3}, i)$ where i is the index of x_{n-2} in

- $\Sigma_q \setminus \{x_{n-5}, x_{n-3}\}$ if $x_{n-5} \neq x_{n-3}$, or
- $\Sigma_q \setminus \{x_{n-4}, x_{n-3}\}$ if $x_{n-5} = x_{n-3}$.

We can prove that $\varphi_1, \varphi_2, \varphi_3$ are bijections as $\varphi_1 \circ \varphi_1^{-1}, \varphi_2 \circ \varphi_2^{-1}, \varphi_2^{-1} \circ \varphi_2, \varphi_3 \circ \varphi_3^{-1}$, and $\varphi_3^{-1} \circ \varphi_3$ are identity maps on their respective sets. Since $\text{Irr}_{\leq 3}(n, q) = \text{Irr}_{\leq 3}^{(s)}(\mathbf{L}_1, n, q) \cup \text{Irr}_{\leq 3}^{(s)}(\mathbf{L}_2, n, q) \cup \text{Irr}_{\leq 3}^{(s)}(\mathbf{L}_3, n, q)$, we have

$$I_{\leq 3}(n, q) = (q-2)I_{\leq 3}(n-1, q) + (q-3)I_{\leq 3}(n-2, q) + (q-2)I_{\leq 3}(n-3, q).$$

□

Corollary 5.4.5. Let $\mathbf{p} \in \Sigma_q^\ell$ For $n \geq \ell + 3$,

$$\begin{aligned} \left| \text{Irr}_{\leq 3}^{(p)}(\mathbf{p}, n, q) \right| &= (q-2) \left| \text{Irr}_{\leq 3}^{(p)}(\mathbf{p}, n-1, q) \right| + (q-3) \left| \text{Irr}_{\leq 3}^{(p)}(\mathbf{p}, n-2, q) \right| \\ &\quad (q-2) \left| \text{Irr}_{\leq 3}^{(p)}(\mathbf{p}, n-3, q) \right|. \end{aligned}$$

5.4.3 Our Conjecture

The results on $k = 2, 3$ provide us a method to compute $\text{rate}_{\leq k}(q)$ for $k \in \{2, 3\}$. In particular, Table 5.1 shows the values of $\text{rate}_{\leq k}(q)$ for $k \in \{2, 3\}$ over certain values of q . Let $T(n, q)$ be the largest size of an $(n, \leq 3; q)$ -TD code and define $\tau(q) \triangleq (1/n) \limsup_{n \rightarrow \infty} \log_q T(n, q)$. It is easy to see that $\text{rate}_{\leq 3}(q) \leq \tau(q) \leq \text{rate}_{\leq 2}(q)$. Therefore, Table 5.1 demonstrates that $\mathcal{C}_{\text{Irr}}(n, \leq 3; q)$ is *almost* optimal for $q \geq 5$. Specifically, the gap between $\text{rate}_{\leq 3}(q)$ and $\text{rate}_{\leq 2}(q)$ is less than 0.01 when $q \geq 5$ and very close to zero when q get larger. We conjecture that $\text{rate}_{\leq 3}(q)$ is indeed the optimal rate for $(n, \leq 3; q)$ -TD code.

q	3	4	5	6	7	8
$\text{rate}_{\leq 2}(q)$	0.4380	0.7249	0.8280	0.8788	0.9081	0.9269
$\text{rate}_{\leq 3}(q)$	0.3479	0.7054	0.8208	0.8753	0.9062	0.9258

Table 5.1: The asymptotic information rates for $\leq k$ -irreducible words for $k \in \{2, 3\}$

5.5 Efficient Encoding/Decoding for $(n, \leq k; q)$ -TD codes, where $k \in \{2, 3\}$

In this section, we look at the encoding/decoding for $(n, \leq k; q)$ -TD codes where $k \in \{2, 3\}$. All known code constructions are based on irreducible words because of two main advantages: *almost optimal rate* and *linear-time decoding*. We first describe an (ℓ, m) -finite state encoder to encode $\leq k$ -irreducible words. For $\epsilon > 0$, we provide a method to choose (ℓ, m) small enough so that the encoder achieves rate that is ϵ away from the optimal. The encoder works in linear time, however, it requires exponential memory space. In order to reduce the size of the requirement space, we provide ranking/unranking algorithms to reduce from exponential to polynomial memory space. While these techniques are standard in constrained coding [3] and combinatorics literature [4], our main contribution is a detailed analysis of the space and time complexities of the respective algorithms.

An input $\mathbf{x}$ is encoded to $\mathbf{y}$ which is an irreducible word and we use this irreducible word in the channel where duplication errors may occur. The idea is illustrated by Figure 5.1. Specifically, we divide input $\mathbf{x}$ into s subwords of length ℓ and encode each subword $\mathbf{x}_i$ to an irreducible word $\mathbf{y}_i$. Finally the encoded word is the concatenation $\mathbf{y}_1 || \mathbf{y}_2 || \dots || \mathbf{y}_s$, an irreducible word of length sm .

Figure 5.1: Encoding/decoding for irreducible words for a duplication error channel

5.5.1 Finite State Encoder

For integers $\ell < m$, an (ℓ, m) -finite state encoder is triple $(\mathcal{S}, \mathcal{E}, \mathcal{L})$, where $\mathcal{S}$ is a set of states, $\mathcal{E} \subset \mathcal{S} \times \mathcal{S}$ is a set of directed edges, and $\mathcal{L} : \mathcal{E} \rightarrow \Sigma_q^\ell \times \Sigma_q^m$ is an edge labeling. To encode $\leq k$ -irreducible words where $k \in \{2, 3\}$, we choose $m \geq 2k - 1$, and set

$$\mathcal{S} \triangleq \text{Irr}_{\leq k}(m, q) \text{ and } \mathcal{E} \triangleq \{(\mathbf{x}, \mathbf{x}') : \mathbf{x}\mathbf{x}' \in \text{Irr}_{\leq k}(2m, q)\}.$$

For $\mathbf{x} \in \mathcal{S}$, we define the *neighbours* of $\mathbf{x}$ to be $N(\mathbf{x}) \triangleq \{\mathbf{x}' : (\mathbf{x}, \mathbf{x}') \in \mathcal{E}\}$. We call the size of $N(\mathbf{x})$ as *degree* of $\mathbf{x}$. The *min degree* of $\mathcal{S}$ is defined as $\Delta_{\mathcal{S}} \triangleq \min\{|N(\mathbf{x})| : \mathbf{x} \in \mathcal{S}\}$. We extend this definition in general, for $\text{Irr}_{\leq k}(m, q)$, $\Delta_{\leq k}(m, q) \triangleq \min\{|N(\mathbf{x})| : \mathbf{x} \in \text{Irr}_{\leq k}(m, q)\}$. We now choose ℓ such that

$$\Delta_{\leq k}(m, q) \geq q^\ell. \tag{5.4}$$

We now define the edge labelling $\mathcal{L}$. For $\mathbf{x} \in \mathcal{S}$, since $|N(\mathbf{x})| \geq q^\ell$, we then use Σ^ℓ to index the first q^ℓ words in $N(\mathbf{x})$. Hence, for $\mathbf{x}' \in S$, if $\mathbf{x}'$ is one of the first q^ℓ words, we let $\mathbf{y}_{\mathbf{x}'} \in \Sigma^\ell$ denote the index. Otherwise, we set $\mathbf{y}_{\mathbf{x}'} = -$. For $(\mathbf{x}, \mathbf{x}') \in \mathcal{E}$, we then set $\mathcal{L}(\mathbf{x}, \mathbf{x}') = (\mathbf{y}_{\mathbf{x}'}, \mathbf{x}')$.

Example 5.5.1. Let $k = 2$, $q = 3$, $m = 3$. Then

$$\begin{aligned} \mathcal{S} &= \{010, 012, 020, 021, 101, 102, 120, 121, 201, 202, 210, 212\} \\ N(010) &= \{201, 210, 212\}, N(012) = \{010, 012, 021, 101, 102\}. \end{aligned}$$

We verify that $\Delta_{\leq 2}(3, 3) = 3$, therefore we choose $\ell = 1$ in order to satisfy constraint 5.4. We may set $\mathcal{L}$ to map the edges exiting the state 010 as follow:

$$(010, 201) \mapsto (0, 201), (010, 210) \mapsto (1, 210), (010, 212) \mapsto (2, 212).$$

We represent the mapping $\mathcal{L}$ using the following lookup table.

$\mathbf{x}$	$N(\mathbf{x})$				
	0	1	2	–	–
010	201	210	212	–	–
012	010	012	021	101	102
020	102	120	121	–	–
021	012	020	021	201	202
101	201	202	210	–	–
102	010	012	101	102	120
120	102	120	121	210	212
121	012	020	021	–	–
201	020	021	201	202	210
202	101	102	120	–	–
210	120	121	201	210	212
212	010	012	021	–	–

Here, to determine $\mathcal{L}(\mathbf{x}, \mathbf{x}')$, we look at the row corresponding to $\mathbf{x}$ and look at the column corresponding to $\mathbf{x}'$. For example, $\mathcal{L}(012, 010) = (0, 010)$.

Encoding: Let s be a positive integer and set $n = s\ell$. Suppose the message $\mathbf{y} = \mathbf{y}_1\mathbf{y}_2\ldots\mathbf{y}_s \in \Sigma^{s\ell}$. To encode $\mathbf{y}$ using an (ℓ, m) -finite state encoder for irreducible words, we do the following:

- (I) Set $\mathbf{x}_0$ to the first word in $\text{Irr}_{\leq k}(m, q)$.
- (II) For $i \in [s]$, set $\mathbf{x}_i$ to be the unique word such that $\mathcal{L}(\mathbf{x}_{i-1}, \mathbf{x}_i) = (\mathbf{y}_i, \mathbf{x}_i)$.
- (III) The encoded irreducible word is $\mathbf{x} = \mathbf{x}_1\mathbf{x}_2\ldots\mathbf{x}_s$.

Example 5.5.2 (Example 5.5.1 continued). Let $s = 3$ and consider the message $\mathbf{y} = 012$. First, we set $\mathbf{x}_0 = 010$. Then $\mathbf{x}_1 = 201$ since $\mathcal{L}(010, 201) = (0, 201)$. Similarly, $\mathbf{x}_2 = 021$ and $\mathbf{x}_3 = 021$. Therefore, the encoded word $\mathbf{x}$ is 201021021.

Observe that the (ℓ, m) -finite state encoder for irreducible words has rate ℓ/m . Since each codeword is of length sm and the size of the code is exactly $q^{s\ell}$, the rate of our encoder as defined in Section 2.2 is

$$\frac{\log_q |\mathcal{C}|}{sm} = \frac{s\ell}{sm} = \frac{\ell}{m}.$$

We now provide a method to choose ℓ and m such that the rate ℓ/m approaches $\text{rate}_{\leq k}(q)$ *quickly*.

Approaching the Asymptotic Information Rate: Pick $\epsilon > 0$. We find suitable values for ℓ and m so that the encoding rate satisfies

$$\ell/m \geq \text{rate}_{\leq k}(q) - \epsilon. \quad (5.5)$$

Recall that ℓ and m are required to satisfy (5.4). Hence, we need to determine $\Delta_{\leq k}(m, q)$ and provide an efficient method to choose ℓ . Surprisingly, these values follow the same recursive equation as $I_{\leq k}(m, q)$.

Proposition 5.5.3. We have that $\Delta_{\leq 2}(3, q) = q(q-2)^2$, $\Delta_{\leq 2}(4, q) = (q-2)^2(q^2 - q - 1)$, and for $m \geq 5$,

$$\Delta_{\leq 2}(m, q) = (q-2)\Delta_{\leq 2}(m-1, q) + (q-2)\Delta_{\leq 2}(m-2, q). \quad (5.6)$$

Proof. Observe that by symmetry, we have $|N(\mathbf{x})| = |N(\mathbf{x}')|$ for $\mathbf{x}, \mathbf{x}' \in \text{Irr}_{\leq 2}^{(s)}(2, m, q)$ or $\text{Irr}_{\leq 2}^{(s)}(3, m, q)$. In addition, we can show that $|N(\mathbf{x})| \leq |N(\mathbf{y})|$ for $\mathbf{x} \in \text{Irr}_{\leq 2}^{(s)}(2, m, q)$ and $\mathbf{y} \in \text{Irr}_{\leq 2}^{(s)}(3, m, q)$. Without loss of generality, assume that $\mathbf{x} \in \text{Irr}_{\leq 2}^{(s)}(010, m, q)$

and $\mathbf{y} \in \text{Irr}_{\leq 2}^{(s)}(210, m, q)$. The neighbours of $\mathbf{x}$ and $\mathbf{y}$ are

$$N(\mathbf{x}) = \left\{ \mathbf{x}' : 10\mathbf{x}' \in \bigcup_{\sigma \notin \{0,1\}} \text{Irr}_{\leq 2}^{(p)}(10\sigma, m+2, q) \right\},$$

$$N(\mathbf{y}) = \left\{ \mathbf{y}' : 10\mathbf{y}' \in \bigcup_{\sigma \neq 0} \text{Irr}_{\leq 2}^{(p)}(10\sigma, m+2, q) \right\}.$$

It is easy to see that $N(\mathbf{x}) \subseteq N(\mathbf{y})$, so the inequality $|N(\mathbf{x})| \leq |N(\mathbf{y})|$ holds. Consequently, $\Delta_{\leq 2}(m, q) = |N(\mathbf{x})|$ where $\mathbf{x} \in \text{Irr}_{\leq 2}^{(s)}(010, m, q)$.

Since $\Delta_{\leq 2}(m, q) = \sum_{\sigma \notin \{0,1\}} |\text{Irr}_{\leq 2}^{(p)}(10\sigma, m+2, q)|$, the recursive equation (5.6) follows from Corollary 5.4.3. $\square$

Proposition 5.5.4. We have that

$$\begin{aligned} \Delta_{\leq 3}(5, q) &= (q-2)(q^2 - 2q - 1)^2, \\ \Delta_{\leq 3}(6, q) &= (q-1)(q^5 - 6q^4 + 9q^3 + 4q^2 - 8q - 9), \\ \Delta_{\leq 3}(7, q) &= (q-2)(q^6 - 6q^4 + 9q^3 + 4q^2 - 8q - 10q + 3), \end{aligned}$$

and for $m \geq 8$,

$$\Delta_{\leq 3}(m, q) = (q-2)\Delta_{\leq 3}(m-1, q) + (q-3)\Delta_{\leq 3}(m-2, q) + (q-2)\Delta_{\leq 3}(m-3, q). \quad (5.7)$$

Proof. We first list out all possible suffixes of length five of an irreducible word, namely $\mathbf{L} = \{abcab, abcac, abcad, abcba, abcbd, abaca, abacb, abacd, abcde, abcdb, abcdc, abcda\}$, where a, b, c, d, e are distinct elements in Σ_q . We first identify which suffix corresponding to words with minimum degree among all the types. We now show that $\Delta_{\leq 3}(m, q) = |N(\mathbf{x})|$ where $\mathbf{x} \in \text{Irr}_{\leq 3}^{(s)}(abcab, m, q)$. In other words, we need to show that $|N(\mathbf{x})| \leq |N(\mathbf{y})|$ for $\mathbf{x} \in \text{Irr}_{\leq 3}^{(s)}(abcab, m, q)$ and $\mathbf{y} \in \text{Irr}_{\leq 3}^{(s)}(\mathbf{p}, m, q)$, where $\mathbf{p} \in \mathbf{L}$. We illustrate our main idea of comparison by one example when $\mathbf{p} = abcad$, while all the remaining cases can be done similarly. Without loss of generality, assume that

$\mathbf{x} \in \text{Irr}_{\leq 3}^{(s)}(01201, m, q)$ and $\mathbf{y} \in \text{Irr}_{\leq 3}^{(s)}(03201, m, q)$. The neighbours of $\mathbf{x}$ and $\mathbf{y}$ are given by

$$N(\mathbf{x}) = \left\{ \mathbf{x}' : 201\mathbf{x}' \in \bigcup_{\sigma \notin \{1,2\}} \text{Irr}_{\leq 3}^{(p)}(201\sigma, m+3, q) \right\},$$

$$N(\mathbf{y}) = \left\{ \mathbf{y}' : 201\mathbf{y}' \in \bigcup_{\sigma \neq 1} \text{Irr}_{\leq 3}^{(p)}(201\sigma, m+3, q) \right\}.$$

Since $N(\mathbf{x}) \subseteq N(\mathbf{y})$, the inequality $|N(\mathbf{x})| \leq |N(\mathbf{y})|$ follows. Hence, $\Delta_{\leq 3}(m, q) = |N(\mathbf{x})|$ where $\mathbf{x} \in \text{Irr}_{\leq 3}^{(s)}(01201, m, q)$.

Since $\Delta_{\leq 3}(m, q) = \sum_{\sigma \notin \{1,2\}} |\text{Irr}_{\leq 3}^{(p)}(201\sigma, m+3, q)|$, the recursive equation (5.7) follows from Corollary 5.4.5. $\square$

We are now ready to present our method to choose ℓ, m efficiently. Recall that λ_2 and λ_3 are roots of the equations $x^2 - (q-2)x - (q-2) = 0$ and $x^3 - (q-2)x^2 - (q-3)x - (q-2) = 0$, respectively. We set κ_2, κ_3 such that $\Delta_{\leq 2}(m, q) \geq \kappa_2 \lambda_2^m$ for $m \in \{3, 4\}$ and $\Delta_{\leq 3}(m, q) \geq \kappa_3 \lambda_3^m$ for $m \in \{5, 6, 7\}$. Using (5.6) and (5.7), we can prove by induction that

$$\Delta_{\leq k}(m, q) \geq \kappa_k \lambda_k^m \text{ for all } m. \quad (5.8)$$

Theorem 5.5.5. Let $k \in \{2, 3\}$. Set $c_k = \text{rate}_{\leq k}(q) = \log_q \lambda_k$. For $\epsilon > 0$, if we choose m and ℓ such that

$$\ell = \left\lceil \frac{(c_k - \epsilon)(c_k - \log_q \kappa_k)}{\epsilon} \right\rceil, \quad (5.9)$$

$$m = \left\lceil \frac{\ell - \log_q \kappa_k}{c_k} \right\rceil, \quad (5.10)$$

then the (ℓ, m) -finite state encoder has rate at least $\text{rate}_{\leq k}(q) - \epsilon$.

Proof. We verify that (5.4) and (5.5) hold for the choice of ℓ and m . Firstly, (5.9) implies that $\epsilon \ell \geq (c_k - \epsilon)(c_k - \log_q \kappa_k)$, and equivalently, $c_k \ell / (\ell - \log_q \kappa_k + c_k) \geq c_k - \epsilon$.

Consequently,

$$\frac{\ell}{m} \geq \frac{\ell}{1 + (\ell - \log_q \kappa_k)/c_k} = \frac{c_k \ell}{\ell - \log_q \kappa_k + c_k} \geq c_k - \epsilon.$$

Hence, we verify (5.5).

Next, from (5.8) and (5.10), we have that

$$\Delta_{\leq k}(m, q) \geq \kappa_k \lambda_k^{(\ell - \log_q \kappa_k)/\log_q \lambda_k} = q^\ell.$$

Hence, we verify (5.4) and complete the proof. $\square$

Running-time analysis. In conclusion, to achieve encoding rates at least $\text{rate}_{\leq k}(q) - \epsilon$, we only require $\ell = \Theta(1/\epsilon)$ and $m = \Theta(1/\epsilon)$. The encoding and decoding algorithms are linear-time complexity, however, if we naively use a lookup table to represent $(\mathcal{S}, \mathcal{E}, \mathcal{L})$, we need $q^{\Theta(1/\epsilon)}$ space. Next, we use combinatorial insights from (5.1) and (5.3) to reduce the space requirement from exponential function as $q^{\Theta(1/\epsilon)}$ to polynomial function as $O(1/\epsilon^2)$.

5.5.2 Ranking/Unranking algorithm

A *ranking function* for a finite set S of cardinality N is a bijection $\text{rank} : S \rightarrow [N]$. Associated with the function rank is a unique *unranking function* $\text{unrank} : [N] \rightarrow S$, such that $\text{rank}(s) = j$ if and only if $\text{unrank}(j) = s$ for all $s \in S$ and $j \in [N]$. In this subsection, we present an algorithm for ranking and unranking $\text{Irr}_{\leq k}(n, q)$. The basis of our ranking and unranking algorithms is the bijections defined in Section 5.4. As implied by the codomains of ϕ and ψ , for $n \geq 4$, we order the words in $\text{Irr}_{\leq k}(n, q)$ such that words in $\text{Irr}_{\leq k}^{(s)}(3, n, q)$ are ordered before words in $\text{Irr}_{\leq k}^{(s)}(2, n, q)$ where $k \in \{2, 3\}$. For words in $\text{Irr}_{\leq k}(2, q)$ and $\text{Irr}_{\leq k}(3, q)$, we simply order them lexicographically. We illustrate the idea behind the unranking algorithm through an example.

Example 5.5.6. Let $n = 6$ and $q = 3$. Then the values of $I_{\leq 2}(m, q)$ are as follow.

m	2	3	4	5	6
$I_{\leq 2}(m, q)$	6	12	18	30	48

Suppose we want to compute $\text{unrank}(40)$. Proposition 5.4.1 gives

$$\text{Irr}_{\leq 2}(6, 3) = \phi(\text{Irr}_{\leq 2}(5, 3) \times [1]) \cup \psi(\text{Irr}_{\leq 2}(4, 3) \times [1]).$$

Now, we are interested in the 40th word of $\text{Irr}_{\leq 2}(6, 3)$. Since $40 > I_{\leq 2}(5, 3) = 30$, the 40th word of $\text{Irr}_{\leq 2}(6, 3)$ is the image of the $40 - 30 = 10$ -th word in $\text{Irr}_{\leq 2}(4, 3)$ under ψ . Recursing tells us that the 10-th word in $\text{Irr}_{\leq 2}(4, 3)$ is the 10-th element in $\phi(\text{Irr}_{\leq 2}(3, 3) \times [1])$. The 10-th element of $\text{Irr}_{\leq 2}(3, 3)$ is 202. This gives

$$\begin{aligned} \text{unrank}(40) &= \psi(\phi(202, 1), 1) \\ &= \psi(202\mathbf{1}, 1) = 2021\mathbf{01}. \end{aligned}$$

The formal unranking algorithm is described in Algorithm 1.

Algorithm 1 $\text{unrank}(n, q, j)$

Require: Integers $n \geq 2$, $q \geq 3$, $1 \leq j \leq I_{\leq 2}(n, q)$

Ensure: $\mathbf{x}$, where $\mathbf{x}$ is the codeword of rank j in $\text{Irr}_{\leq 2}(n, q)$

if $n \leq 3$ **then**

return j -th codeword in $\text{Irr}_{\leq 2}(n, q)$

if $j \leq (q - 2)I_{\leq 2}(n - 1, q)$ **then**

$j' \leftarrow 1 + \lfloor (j - 1) / (q - 2) \rfloor$

$i \leftarrow (j - 1) \pmod{q - 2} + 1$

return $\phi(\text{unrank}(n - 1, q, j'), i)$

else

$j' \leftarrow 1 + \lfloor (j - (q - 2)I_{\leq 2}(n - 1, q) - 1) / (q - 2) \rfloor$

$i \leftarrow (j - (q - 2)I_{\leq 2}(n - 1, q) - 1) \pmod{q - 2} + 1$

return $\psi(\text{unrank}(n - 2, q, j'), i)$

The ranking algorithm for $\text{Irr}_{\leq 2}(n, q)$ is described similarly in Algorithm 2.

Example 5.5.7. Let $n = 6$ and $q = 3$ as before. Suppose we want to compute $\text{rank}(202101)$. Since $202101 \in \text{Irr}_{\leq 2}^{(s)}(2, 6, 3)$, we have that 202101 is obtained from

applying ψ to $2021 \in \text{Irr}_{\leq 2}(4, 3)$. Again, since $2021 \in \text{Irr}_{\leq 2}^{(s)}(3, 6, 3)$, we have that 202 is obtained from applying ϕ to $202 \in \text{Irr}_{\leq 2}(3, 3)$. Therefore,

$$\begin{aligned} \text{rank}(202101) &= \text{rank}(2021) + I_{\leq 2}(5, 3) \\ &= \text{rank}(202) + I_{\leq 2}(5, 3) \\ &= 10 + 30 = 40 \end{aligned}$$

Algorithm 2 $\text{rank}(n, q, \mathbf{x})$

Require: $n \geq 2$, $q \geq 3$ and irreducible word $\mathbf{x}$ of length n

Ensure: j , where $1 \leq j \leq I_{\leq 2}(n, q)$, the rank of $\mathbf{x}$ in $\text{Irr}_{\leq 2}(n, q)$

```

if  $n \leq 3$  then
  return  $\text{rank}(\mathbf{x})$  in  $\text{Irr}_{\leq 2}(n, q)$ 
if  $x_n \neq x_{n-2}$  then
   $\mathbf{x}' \leftarrow x_1 x_2 \dots x_{n-1}$ 
   $i \leftarrow$  the index of  $x_n$  in  $\Sigma_q \setminus \{x_{n-2}, x_{n-1}\}$ 
  return  $(\text{rank}(n-1, q, \mathbf{x}') - 1)(q-2) + i$ 
else
   $\mathbf{x}' \leftarrow x_1 x_2 \dots x_{n-2}$ 
   $i \leftarrow$  the index of  $x_{n-1}$  in  $\Sigma_q \setminus \{x_{n-3}, x_{n-2}\}$ 
  return  $(\text{rank}(n-2, q, \mathbf{x}') - 1)(q-2) + i + (q-2)I_{\leq 2}(n-1, q)$ 

```

The set of values of $\{I_{\leq 2}(m, q) : m \leq n\}$ required in Algorithms 1 and 2 can be precomputed based on the recurrence (5.1). Since the numbers $I_{\leq 2}(m, q)$ grow exponentially, these n stored values require $O(n^2)$ space.

Running-time analysis: Algorithms 1 and 2 involve $O(n)$ iterations and each iteration involves a constant number of arithmetic operations. Therefore, Algorithms 1 and 2 involve $O(n)$ arithmetics operations and have time complexity $O(n^2)$. Similarly, we present out ranking/unranking algorithm for $\text{Irr}_{\leq 3}(n, q)$ in Algorithm 3 and 4.

5.5.3 Reducing the Space Requirement for the Finite State Encoder

As discussed earlier, a naive implementation of the (ℓ, m) -finite state encoder requires $q^{\Theta(m)}$ space (assuming $\ell = \Theta(m)$). Here, we modify our unranking algorithm to

Algorithm 3 $\text{unrank}(n, q, j)$

Require: Integers $n \geq 3$, $q \geq 3$, $1 \leq j \leq I_{\leq 3}(n, q)$

Ensure: $\mathbf{x}$, where $\mathbf{x}$ is the codeword of rank j in $\text{Irr}_{\leq 3}(n, q)$

```
if  $n \leq 5$  then
  return  $j$ -th codeword in  $\text{Irr}_{\leq 3}(n, q)$ 
if  $j \leq (q-2)I_{\leq 3}(n-1, q)$  then
   $j' \leftarrow 1 + \lfloor (j-1)/(q-2) \rfloor$ 
   $i \leftarrow (j-1) \pmod{q-2} + 1$ 
  return  $\varphi_1(\text{unrank}(n-1, q, j'), i)$ 
else
   $j' \leftarrow j - (q-2)I_{\leq 3}(n-1, q)$ 
  if  $j' \leq (q-3)I_{\leq 3}(n-2, q)$  then
     $j' \leftarrow 1 + \lfloor (j'-1)/(q-3) \rfloor$ 
     $i \leftarrow (j'-1) \pmod{q-3} + 1$ 
    return  $\varphi_2(\text{unrank}(n-2, q, j'), i)$ 
  else
     $j' \leftarrow j' - (q-3)I_{\leq 3}(n-2, q)$ 
     $i \leftarrow (j'-1) \pmod{q-2} + 1$ 
    return  $\varphi_3(\text{unrank}(n-3, q, j'), i)$ 
```

Algorithm 4 $\text{rank}(n, q, \mathbf{x})$

Require: $n \geq 3$, $q \geq 3$ and irreducible word $\mathbf{x}$ of length n

Ensure: j , where $1 \leq j \leq I_{\leq 3}(n, q)$, the rank of $\mathbf{x}$ in $\text{Irr}_{\leq 3}(n, q)$

```
if  $n \leq 5$  then
  return  $\text{rank}(\mathbf{x})$  in  $\text{Irr}_{\leq 3}(n, q)$ 
if  $\mathbf{x} \in \text{Irr}_{\leq 3}^{(s)}(\mathbf{L}_1, n, q)$  then
   $(\mathbf{x}', i) \leftarrow \varphi_1^{-1}(\mathbf{x})$ 
  return  $(\text{rank}(n-1, q, \mathbf{x}') - 1)(q-2) + i$ 
if  $\mathbf{x} \in \text{Irr}_{\leq 3}^{(s)}(\mathbf{L}_2, n, q)$  then
   $(\mathbf{x}', i) \leftarrow \varphi_2^{-1}(\mathbf{x})$ 
  return  $(\text{rank}(n-2, q, \mathbf{x}') - 1)(q-3) + i + (q-2)I_{\leq 3}(n-1, q)$ 
if  $\mathbf{x} \in \text{Irr}_{\leq 3}^{(s)}(\mathbf{L}_3, n, q)$  then
   $(\mathbf{x}', i) \leftarrow \varphi_3^{-1}(\mathbf{x})$ 
  return  $(\text{rank}(n-3, q, \mathbf{x}') - 1)(q-2) + i + (q-2)I_{\leq 3}(n-1, q) + (q-3)I_{\leq 3}(n-2, q)$ 
```

reduce the space requirement $O(m)$ integers or $O(m^2)$ bits. In particular, let $\mathbf{x}_{i-1} \in \text{Irr}_{\leq 2}(m, q)$ and $\mathbf{y}_i \in \Sigma_q^\ell$. Our encoding task is to determine the irreducible word $\mathbf{x}_i$ in $N(\mathbf{x}_i)$ whose index corresponds to $\mathbf{y}_i$. Equivalently, if j is the rank of $\mathbf{y}_i \in \Sigma_q^\ell$, then our task is to find $\mathbf{x}_i$ such that its rank in $N(\mathbf{x}_{i-1})$ is j . Since $\mathbf{x}_{i-1}$ is irreducible and using symmetry, we assume that $\mathbf{x}_{i-1} \in \text{Irr}_{\leq 2}^{(s)}(010, m, q)$ or $\mathbf{x}_{i-1} \in \text{Irr}_{\leq 2}^{(s)}(210, m, q)$. Furthermore, from Proposition 5.5.4 we see that $N(\mathbf{x}_{i-1})$ corresponds to a union of ≤ 2 -irreducible words with prefixes of the form 10σ . Therefore, it suffices to provide ranking/unranking algorithms for $\text{Irr}_{\leq 2}^{(p)}(10\sigma, m, q)$.

Since (5.2) implies that $\text{Irr}_{\leq 2}^{(p)}(10\sigma, m, q)$ has the same recursive structure as $\text{Irr}_{\leq 2}(m, q)$, we can modify Algorithms 1 and 2 to unrank and rank $\text{Irr}_{\leq 2}^{(p)}(10\sigma, m, q)$.

Now, to rank/unrank $\text{Irr}_{\leq 2}^{(p)}(10\sigma, m, q)$ require $O(m)$ precomputed integers. Assuming q is constant, we require only $O(m)$ integers or $O(m^2)$ bits. However, the running time is increased to $O(m^2)$.

5.6 The First Known Solution when $k > 3$

When $k \geq 4$, there is no known result on $(n, \leq k; q)$ -TD codes. The set of all irreducible words does not form a code. There are examples of different irreducible words that share a common descendant. For example, when $q = 3, k = 4$, two words $\mathbf{x} = 012$ and $\mathbf{y} = 0121012$, which are two irreducible words in $\text{Irr}_{\leq 4}(3)$, share a common descendant $\mathbf{z} = 012101212$.

$$\begin{aligned} 012 &\xrightarrow[2]{*} 012\textcolor{red}{12} \xrightarrow[4]{*} 0121\textcolor{red}{0121}2 \text{ and} \\ 0121012 &\xrightarrow[2]{*} 0121012\textcolor{red}{12}. \end{aligned}$$

To design $(n, \leq k; q)$ -TD codes over $\Sigma = \{0, 1, \dots, q-1\}$, we make use of the condition where each tandem duplication is of length at most k . The first trivial construction is to concatenate different congruent classes modulo $(k+1)$.

Construction 5.6.1. Assume that $k + 1 \leq q$, $q = (k + 1)r$ and $n = (k + 1)t$. We partition the set of symbols Σ into $(k + 1)$ congruent classes $\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_{k+1}$ modulo $(k + 1)$. In other words, $\mathcal{S}_i = \{j \in \Sigma : j \equiv i \pmod{k + 1}\}$, and $|\mathcal{S}_i| = r$ where $1 \leq i \leq k + 1$. We construct code $\mathcal{C} = \mathcal{C}_1 || \mathcal{C}_2 || \dots || \mathcal{C}_t$ where each codeword $\mathbf{c} = (c_1, c_2, \dots, c_n) \in \mathcal{C}$, $\mathbf{c} = \mathbf{c}_1 || \mathbf{c}_2 || \dots || \mathbf{c}_t$ where $\mathbf{c}_i \in \mathcal{C}_i$ of length $(k + 1)$, $\mathbf{c}_i = (c_{(i-1)(k+1)+1}, c_{(i-1)(k+1)+2}, \dots, c_{i(k+1)}) \in \Sigma^{k+1}$ such that $c_{(i-1)(k+1)+j} \in \mathcal{S}_j$ for $1 \leq j \leq k + 1$.

Theorem 5.6.2. The constructed code $\mathcal{C}$ is a $(n, \leq k; q)$ -TD code of size $(\frac{q}{k+1})^n$, and thus the rate is $1 - \log_q(k + 1)$.

Proof. We prove the correctness of Construction 5.6.1 by providing a decoding algorithm. Assume that some tandem duplications occur in a codeword $\mathbf{c} = (c_1, c_2, \dots, c_n) \in \mathcal{C}$ resulting a word $\mathbf{x}$. We now provide an efficient decoding algorithm to recover the original codeword $\mathbf{c}$. We observe that each duplication is of length at most k , and each codeword is constructed as the concatenation of $k + 1$ different congruent classes. Therefore, the distance between two symbols that belong to the same congruent class in each codeword must be always $k + 1$. Assume that $\mathbf{c} = \mathbf{c}_1 || \mathbf{c}_2 || \dots || \mathbf{c}_t$ where $\mathbf{c}_i \in \mathcal{C}_i$. Our decoding algorithm has t steps.

Step 1. We first recover $\mathbf{c}_1 = (c_1, c_2, \dots, c_{k+1})$ by searching for the first $k + 1$ symbols that belong to $\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_{k+1}$ respectively in $\mathbf{x}$. This is because the order of the first appearance of $c_1, c_2, \dots, c_{k+1}$ is preserved under tandem duplications. We mark down the position of the first appearance of c_{k+1} and continue to recover the next $(k + 1)$ symbols from this position.

Step i . We recover the subword $\mathbf{c}_i = (c_{(i-1)(k+1)+1}, c_{(i-1)(k+1)+2}, \dots, c_{i(k+1)})$, where $2 \leq i \leq t$ with the assumption that we already recovered $\mathbf{c}_{i-1}$ and our marker is at the position of symbol $c_{(i-1)(k+1)}$. Again, these $(k + 1)$ symbols in $\mathbf{c}_i$ belonging to different congruent classes $\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_{k+1}$ respectively after $c_{(i-1)(k+1)}$. We search for the first appearance of symbols in $\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_{k+1}$ in order after $c_{(i-1)(k+1)}$. At the end

of this step, we successfully recover $\mathbf{c}_i = (c_{(i-1)(k+1)+1}, c_{(i-1)(k+1)+2}, \dots, c_{(i-1)(k+1)+k+1})$ and our marker is at the position of symbol $c_{i(k+1)}$.

Since the size of each set $\mathcal{S}_i$ is $q/(k+1)$, the size of constructed code is $(\frac{q}{k+1})^n$. $\square$

Example 5.6.3. Let $n = 6, q = 6, k = 2, \mathcal{S}_1 = \{0, 3\}, \mathcal{S}_2 = \{1, 4\}, \mathcal{S}_3 = \{2, 5\}$. Suppose that we receive a vector $\mathbf{x} = 01012123231315$. Firstly, we search for the first three symbols in different congruent classes in $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$, respectively in $\mathbf{x} = 01012123231315$. We then find the next three symbols in $\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3$, respectively after the last recovered symbol in $\mathbf{x} = 01012123231315$. Thus, the original codeword is $\mathbf{c} = 012315$.

We now show that we can do much better than $(\frac{q}{k+1})^n$.

Construction 5.6.4. For $n, q, k \in \mathbb{Z}, k+1 \leq q, n = mk$ for some $m \in \mathbb{Z}$, we construct code $\mathcal{C} = \mathcal{C}_1 || \mathcal{C}_2 || \dots || \mathcal{C}_m$ where each codeword $\mathbf{c} = (c_1, c_2, \dots, c_n) \in \mathcal{C}$, $\mathbf{c} = \mathbf{c}_1 || \mathbf{c}_2 || \dots || \mathbf{c}_m$ where $\mathbf{c}_i \in \mathcal{C}_i$ of length k , $\mathbf{c}_i = (c_{(i-1)k+1}, c_{(i-1)k+2}, \dots, c_{ik}) \in \Sigma^k$ satisfies the following conditions:

1. $c_{(i-1)k+s} \neq c_{(i-1)k+t}$ for distinct $s, t \in [k]$, and
2. For $i \geq 2$, the first symbol, $c_{(i-1)k+1}$ is different from k symbols in previous subword $\mathbf{c}_{i-1}$.

Theorem 5.6.5. The codes constructed is capable of correcting tandem duplication of length up to k . The code's size is at least

$$\left((q-1)(q-2) \cdots (q-k) \right)^{n/k},$$

with the asymptotic rate is $\frac{\log_q(q-1) + \log_q(q-2) + \dots + \log_q(q-k)}{k}$.

Proof. It is easy to see that the size of $\mathcal{C}_1$ is $q(q-1) \cdots (q-k+1)$ and the size of $\mathcal{C}_i$,

for $2 \leq i \leq m$, is $(q-1) \cdots (q-k)$. Therefore, the size of constructed code is

$$(q(q-1) \cdots (q-k+1))((q-1) \cdots (q-k))^{k-1} \geq ((q-1) \cdots (q-k))^k$$

We establish the theorem by giving an efficient decoding algorithm that recovers each codeword from a sequence of tandem duplications of length at most k . Suppose that some tandem duplications occur in codeword $\mathbf{c} = (c_1, c_2, \dots, c_n) \in \mathcal{C}$ resulting a word $\mathbf{x}$. Assume that $\mathbf{c} = \mathbf{c}_1 || \mathbf{c}_2 || \cdots || \mathbf{c}_m$. Our decoding algorithm has m steps.

Step 1. We first recover $\mathbf{c}_1 = (c_1, c_2, \dots, c_k)$. We search for the first k distinct symbols in $\mathbf{y}$. In order of appearance, the symbols are $c_1, c_2, \dots, c_k$. This is because the order of the first appearance of $c_1, c_2, \dots, c_k$ is preserved under tandem duplications. We mark down the position of the first appearance of c_k .

Step 2. We now recover $\mathbf{c}_2 = (c_{k+1}, c_{k+2}, \dots, c_{2k})$. Observe that $\mathbf{c}_1$ is recovered. From the position of our marker, we continue to search for the first symbol that is different from the previous k recovered symbols. Such symbol is c_{k+1} , which is the first symbol in $\mathbf{c}_2$. Our marker now changes to the position when we find c_{k+1} . Next, from the position of our marker, we continue to find the next $(k-1)$ distinct symbols in order. During this process, at some position i , if c_{k+1} appears again, that means certain duplication errors happen. We delete all symbols between our marker and the symbol at position i . We then restart this process until we find the next $k-1$ distinct symbols in order after the position of our marker. At the end of step 2, we successfully recover $\mathbf{c}_2$ and our marker changes to the position when we find c_{2k} .

Step i. We recover $\mathbf{c}_i = (c_{(i-1)k+1}, c_{(i-1)k+2}, \dots, c_{ik})$. We recover $\mathbf{c}_i$ with the assumption that $\mathbf{c}_{i-1}$ has been recovered and our marker is at $c_{(i-1)k}$. Similar to step 2, from the position of our marker, we search for the first symbol that is different from those k symbols in $\mathbf{c}_{i-1}$. It would be $c_{(i-1)k+1}$, the first symbol in $\mathbf{c}_i$. Our marker now changes to the position when we find $c_{(i-1)k+1}$. From the position of our marker, we search for the next $(k-1)$ distinct symbols in order followed by the same rule as described in

step 2. At the end of this step, we successfully recover $\mathbf{c}_i$ and our marker changes to the position when we find c_{ik} . $\square$

Example 5.6.6. Let $n = 8, q = 8, k = 4, m = 2$. Each codeword $\mathbf{c} \in \mathcal{C}$ is the concatenation of $\mathbf{c}_1$ and $\mathbf{c}_2$ of length four. Suppose that we receive a vector $\mathbf{x} = 2121681816846848468282$.

Step 1: recover $\mathbf{c}_1$. The first four distinct symbols in $\mathbf{x}$ are highlight in red.

$$\mathbf{x} = \textcolor{red}{212168}1816846848468282.$$

Hence $\mathbf{c}_1 = \textcolor{red}{2168}$.

Step 2: recover $\mathbf{x}_2$.

- To recover the first symbol in $\mathbf{c}_2$, we search for the first symbol that is different from $\{2, 1, 6, 8\}$:

$$\mathbf{x} = \textcolor{red}{212168}18168\textcolor{blue}{4}6848468282.$$

Hence, symbol $\textcolor{blue}{4}$ is the first symbol in $\mathbf{c}_2$.

- To recover the remaining symbols in $\mathbf{c}_2$, we search for the next three distinct symbols after symbol $\textcolor{blue}{4}$:

$$\mathbf{x} = \textcolor{red}{212168}18168\textcolor{blue}{468}282.$$

Hence, $\mathbf{c}_2 = \textcolor{blue}{4682}$.

Thus, the original codeword is $\mathbf{c} = \textcolor{red}{2168}\textcolor{blue}{4682}$.

Each codeword in $\mathcal{C}$ is an $\leq k$ -irreducible word. Hence, our constructed code is a subset of irreducible words that forms a code. When $k = 4$, the rate of constructed code is at least $1/4 \times (\log_q(q-1) + \log_q(q-2) + \log_q(q-3) + \log_q(q-4))$ which is very

close to the upper bound. Table 5.2 demonstrates that $(n, \leq 4; q)$ -TD code is *almost* optimal for large q .

q	Upper Bound	Lower Bound
10	0.9542	0.8701
15	0.9745	0.9311
20	0.9828	0.9547
25	0.9873	0.9668
30	0.9900	0.9741
35	0.9918	0.9789

Table 5.2: Upper and Lower Bounds on $(n, \leq 4; q)$ -TD codes

Chapter 6

Conclusion and Future Direction

Throughout this thesis, we focus on designing codes correcting burst errors with efficient error decoders. Specifically, we construct burst deletions/insertions-correcting codes over permutations, multipermutations, or generally over q -ary alphabet of size more than two. We establish the correctness of our constructions by providing efficient error decoders to recover the original codewords in linear time.

In permutations, deletions and insertions are classified into two types as stable and unstable. For a fixed integer s , our designed codes against a burst of s stable deletions/insertions have size at least $\Theta(n!/n^2)$ while the upper bound is $\Theta(n!/n)$. This size is much larger than trivial interleaved codes as described in 3.3.2. On the other hand, we construct a family of permutation codes correcting a burst of s unstable deletions/insertions have size at least $\left(\frac{(n/s)!}{(n/s)^2}\right)^s$. Even though this size is not close to the upper bound that was derived in [19], the rate is still asymptotically one for fixed integer s . In general, codes correcting unstable errors are harder to deal with than codes correcting stable errors. There remains a big gap between our constructive lower bound and upper bound. Designing such codes with size closer to the upper bound is a subject of further research.

In multipermutations or q -ary constant-composition codes, we construct a family of codes capable of correcting a burst of s deletions/insertions where the sizes are at least $\Theta(1/n)$ of the upper bound. Currently, in permutations and multipermutations, we have no results on efficient algorithms for channel encoding or source decoding. In literature, there are many works on efficient encoding/decoding for binary deletion/insertion channel [6–8], however, the research on an alphabet of size more than two is very limited. Another future direction is to design efficient channel encoders/source decoders for permutations/multipermutations or q -ary deletion/insertion channel with the purpose of achieving high rate and low complexity.

In Chapter 5, we construct codes correcting tandem duplications (TD), a special type of insertion errors. Such codes have attracted recent attention due to the applications that store data in living organisms. Besides code constructions, we also look at efficient encoding/decoding algorithms for channel encoder and source decoder. While most of previous works are to deal with duplications of length less than or equal to three, we provide the first known solution to construct codes combatting duplications of length more than three. Our work mainly focused on *unbounded* TD codes, it could be interesting to consider *bounded* TD codes. In other words, we assume that the number of duplication errors that can occur is bounded above by a number N , and an $(n, N, \leq k, q)$ -TD code is a code that can correct up to N tandem duplications of length at most k over an alphabet of size q . Designing such codes with optimal sizes is a subject of further research.

Bibliography

- [1] Shannon, “C. E. A mathematical theory of communication”, *Bell System Tech. J.* 27 (1948), 379-423.
- [2] Y. M. Chee and S. Ling, “Constructions for q-Ary Constant-Weight Codes”, *IEEE Trans. Inform. Theory*, vol. 53, no. 1, pp. 135-146, 2007.
- [3] B. H. Marcus, R. M. Roth, and P. H. Siegel, “An Introduction to Coding for Constrained System”, Oct 2001.
- [4] A. Nijenhuis, and H. S. Wilf, *Combinatorial Algorithms: for Computers and Calculators*, Elsevier, 2014.
- [5] J. Konc, and D. Janezic, “An improved branch and bound algorithm for the maximum clique problem”, *MATCH Commun. Math. Comput. Chem.*, 2007, vol. 58, pp. 569-590, <http://www.sicmm.org/konc/maxclique>
- [6] K. A. S. Abdel-Ghaffar and H. C. Ferreira, “Systematic encoding of the Varshamov-Tenengolts codes and the Constantin-Rao Codes”, *IEEE Trans. Inform. Theory*, vol. 44, pp. 340-345, Jan 1998.
- [7] V. Guruswami and C. Wang, “Deletion codes in the high-noise and high-rate regimes”, *IEEE Trans. Inform. Theory*, vol. 63, no. 4, pp. 1961-1970, 2017.
- [8] J. Brakensiek, V. Guruswami, S. Zbarsky, “Efficient Low-Redundancy Codes for Correcting Multiple Deletions”, *SODA*, 2016.

- [9] V. Guruswami, R. Li, “Efficiently Decodable Codes for the Binary Deletion Channel”, *RANDOM*, 2017.
- [10] N. J. A. Sloane, “On single-deletion-correcting codes”, in *Codes and Designs, Ohio State University (Ray-Chaudhuri Festschrift)*, pp. 273-291, 2000.
- [11] E. A. Ratzer, “Marker codes for channels with insertions and deletions”, *Ann. Telecommun.*, vol. 60, no. 1, pp. 29-44, 2005.
- [12] Z. Liu and M. Mitzenmacher, “Codes for deletion and insertion channels with segmented errors,” *IEEE Trans. Inf. Theory*, vol. 56, no. 1, pp. 224-232, 2010.
- [13] S. M. S. Tabatabaei Yazdi and L. Dolecek, “Synchronization from deletions through interactive communication,” *IEEE Trans. Inf. Theory*, vol. 60, pp. 397-409, Jan. 2014.
- [14] F. Sala, C. Schoeny, N. Bitouze, and L. Dolecek, “Synchronizing files from a large number of insertions and deletions,” *IEEE Trans. Commun.*, vol. 64, no. 6, pp. 2258-2273, 2016.
- [15] M. Abroshan, R. Venkataramanan, and A. Guillen i Fabregas, “Coding for segmented edit channels,” *IEEE Trans. Inf. Theory*, vol. 64, no. 4, pp. 3086-3098, 2017.
- [16] K. A. Abdel-Ghaffar, F. Paluncic, H. C. Ferreira, and W. A. Clarke, “On Helberg’s generalization of the Levenshtein code for multiple deletion/insertion error correction”, *IEEE Trans. Inf. Theory*, vol. 58, no. 3, pp. 1804-1808, 2012.
- [17] S. K. Hanna and S. E. Rouayheb, “Guess check codes for deletions and synchronization”, in *ISIT 2017-Proceedings of the 2017 IEEE International Symposium on Information Theory*, 2017.
- [18] V. I. Levenshtein, “On perfect codes in deletion and insertion metric”, *Discrete Math. Appl.*, 3:241-258, 1992.

- [19] R. Gabrys, E. Yaakobi, F. Farnoud, F. Sala, J. Bruck, and L. Dolecek, “Codes correcting erasures and deletions for rank modulation”, *IEEE Trans. Inform. Theory*, 62(1):136-150, 2016.
- [20] D. Slepian, “Permutation modulation”, *Proc. IEEE*, vol. 53, pp. 228-236, Mar. 1965.
- [21] A. J. Han Vinck, “Coded modulation for power line communications”, *AEU Journal*, pp. 45-49, January 2000.
- [22] A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck, “Rank modulation for flash memories”, *IEEE Trans. Inform. Theory*, 55(6):2659-2673, 2009.
- [23] A. Jiang, M. Schwartz, and J. Bruck, “Correcting charge constrained errors in the rank-modulation scheme”, *IEEE Trans. Inform. Theory*, 56(5):2112-2120, 2010.
- [24] E. En. Gad, M. Langberg, M. Schwartz, and J. Bruck, “Constant-weight Grays codes for local rank modulation”, *IEEE Trans. Information Theory*, vol. 57, pp. 7431-7442, Nov. 2011.
- [25] H. Zhou, M. Schwartz, A. Jiang, and J. Bruck, “Systematic error correcting codes for rank modulation”, *IEEE Trans. Inform. Theory*, 61(1):17-32, 2015.
- [26] E. En Gad, A. Jiang, and J. Bruck, “Trade-offs between instantaneous and total capacity in multi-cell flash memories”, in *ISIT 2012-Proceedings of the 2012 IEEE International Symposium on Information Theory*, Cambridge, MA, USA, Jul. 2012, pp. 704-708.
- [27] J.-D. Lee, S.-H. Hur, and J.-D. Choi, “Effects of floating-gate interference on nand flash memory cell operation”, *IEEE Electron. Device Lett.*, 23(5):264-266, 2002.
- [28] K. Prall, “Scaling non-volatile memory below 30nm”, in *22nd IEEE Non-Volatile Semiconductor Memory Workshop*, 2007.

- [29] V. I. Levenshtein, “Binary codes capable of correcting deletions, insertions, and reversals”, *Soviet Physics Dokl.*, 10:707-710, 1965.
- [30] G. Tenengolts, “Nonbinary codes, correcting single deletion or insertion”, *IEEE Trans. Inform. Theory*, 30(5): 766-769, 1984.
- [31] C. Schoeny, A. Wachter, R. Gabrys, and E. Yaakobi, “Codes correcting a burst of deletions or insertions”, in *ISIT 2016-Proceedings of the 2016 IEEE International Symposium on Information Theory*, pp 630-634, 2016.
- [32] R. R. Varshamov and G. M. Tenengolts, “Codes which correct single asymmetric errors (in russian)”, *Automatika i Telemekhanika*, vol. 161, no. 3, pp.288-292, 1965.
- [33] L. Cheng, T. G. Swart, H. C. Ferreira and K. A. S. Abdel-Ghaffar, “Codes for Correcting Three or More Adjacent Deletions or Insertions”, in *ISIT 2014-Proceedings of the 2014 IEEE International Symposium on Information Theory*, 2014.
- [34] Y. M. Chee, V. K. Vu, and X. Zhang, “Permutation codes correcting a single burst deletion I: unstable deletions”, in *ISIT 2015-Proceedings of the 2015 IEEE International Symposium on Information Theory*, pp 1741-1745, 2015.
- [35] Y. M. Chee, S. Ling, T. T. Nguyen, V. K. Vu, and H. Wei, “Permutation codes correcting a single burst deletion II: stable deletions”, in *ISIT 2017-Proceedings of the 2017 IEEE International Symposium on Information Theory*, 2017.
- [36] S. Yazdi, H. M. Kiah, E. R. Garcia, J. Ma, H. Zhao, and O. Milenkovic, “DNA-based storage: Trends and methods,” *IEEE Trans. Molecular, Biological, Multi-Scale Commun.*, vol. 1, no. 3, pp. 230-248, 2015.
- [37] P. Leupold, V. Mitrana, and J. M. Sempere, “Formal Languages Arising from Gene Repeated Duplication”, *Aspects of Molecular Computing*, Springer, 2004, pp. 297-308.

- [38] P. Leupold, C. Martin-Vide, and V. Mitrana, “Uniformly bounded duplication languages”, *Discrete Applied Mathematics*, vol. 146, no. 3, pp 301-310, 2005.
- [39] E. S. Lander, L. M. Linton, B. Birren, C. Nusbaum, M. C. Zody, J. Baldwin, K. Devon, K. Dewar, M. Doyle, W. FitzHugh et al., “Initial sequencing and analysis of the human genome”, *Nature*, vol. 409, no. 6822, pp. 860-921, 2001.
- [40] G. R. Sutherland and R. I. Richards, “Simple tandem DNA repeats and human genetic disease”, *Proceedings of the National Academy of Sciences*, vol. 92, no. 9, pp. 3636-3641, 1995.
- [41] J. W. Fondon and H. R. Garner, “Molecularoriginsofrapidandcontinuousmorphological evolution”, *Proceedings of the National Academy of Sciences*, vol. 101, no. 52, pp. 18058-18063, 2004.
- [42] M. Arita and Y. Ohashi, “Secret signatures inside genomic DNA”, *Biotechnology progress*, vol. 20, no. 5, pp. 1605-1607, 2004.
- [43] D. Heider and A. Barnekow, “DNA-based watermarks using the DNA-Crypt algorithm”, *BMC Bioinformatics*, vol. 8, no. 1, p. 176, 2007.
- [44] M. Liss, D. Daubert, K. Brunner, K. Kliche, U. Hammes, A. Leiherer, and R. Wagner, “Embedding permanent watermarks in synthetic genes”, *PloS one*, vol. 7, no. 8, p. e42465, 2012.
- [45] 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.
- [46] S. Jain, F. Farnoud, and J. Bruck, “Capacity and Expressiveness of Genomic Tandem Duplication”, *IEEE International Symposium on Information Theory*, 2015.