CORRECTING MULTIPLE BURSTS OF EXACTLY t DELETIONS

Zhuoer Shen

Charlottesville, VA

Bachelor of Arts, University of Virginia, 2023

A Thesis submitted to the Graduate Faculty

of the University of Virginia in Candidacy for the Degree of

Master of Science

Department of Computer Science

University of Virginia

Feburary 2025

Prof. Wei-Kai Lin, Chair

Prof. Farzad Farnoud

Prof. Chen-Yu Wei

Correcting Multiple Bursts of Exactly t Deletions

Zhuoer Shen

(ABSTRACT)

Stored or transmitted data is often subject to errors arising from noise, interference, or physical defects. Error-correcting codes play a crucial role in detecting and recovering from such errors to ensure reliable communication and data storage. In this thesis, we focus on burst deletion errors, where multiple consecutive symbols may be removed from a sequence. We propose novel code constructions that efficiently correct multiple bursts of exactly t deletions. Our primary contribution is a generalizable framework that extends existing deletion-correcting methods to handle multiple bursts in both binary and q-ary sequences.

Specifically, we construct a code capable of correcting two bursts of exactly t deletions in binary sequences with redundancy at most $6.4 \log n + 10 \log \log n + O(1)$. We further extend our construction to q-ary sequences, achieving redundancy $7.4 \log n + 10 \log \log n + 3 \log q + O(1)$. Additionally, we introduce a code for three bursts of exactly t deletions with redundancy at most $15 \log n + o(\log n)$, and an encoding and decoding complexity of $O(n^7)$.

Our approach combines deletion-correcting codes, such as Guruswami and Håstad's explicit two-deletion code, with efficient erasure-correcting codes, including Dumer's linear codes with designed distance 5. For a code correcting three bursts of exactly t deletions, we employ syndrome compression to identify error locations and leverage

structured redundancy to improve efficiency. We analyze the time complexity of our encoding and decoding process and show that our three bursts of t deletion correcting code has an encoding and decoding complexity of $O(n^7)$, which is asymptotically more efficient than a direct application of syndrome compression for large t.

This work contributes to the broader study of deletion-correcting codes by providing explicit constructions that balance redundancy, efficiency, and scalability. The proposed techniques offer promising directions for improving deletion error resilience in DNA storage, document synchronization protocols, and digital communications. We conclude with potential extensions, including the correction of multiple bursts of 2 erasures and further optimization of redundancy in multi-burst scenarios.

Dedication

In loving memory of my mother

To my loving family

For all your support

Acknowledgments

I am truly thankful for the continuing support, encouragement, and help of many individuals and organizations. My family provided me with much help and emotional support. I dedicate this thesis to them all and thank them for giving me the support that I needed to go through this journey.

I am grateful for the support, invaluable advice, and encouragement from my advisor, Prof. Farzad Farnoud. Prof. Farnoud expects and demands the best from his students, and I strove to live up to those expectations. I also want to extend my sincere appreciation to Yuting Li for his continuous guidance, encouragement, and constructive advice. Yuting provided me with invaluable insight into the background knowledge and the mathematics used in this research.

A great thank you also goes to the other members of my dissertation committee: Prof. Wei-Kai Lin, and Prof. Chen-Yu Wei, for their guidance on my thesis supervisory committee.

Contents

Chapter 1

Introduction

1.1 Motivation and Background

In modern communication systems, error correction is a cornerstone for ensuring reliable data transmission in the presence of noise and interference. Among the various types of errors, deletion errors, particularly burst deletions, pose a significant challenge. Burst deletions occur when consecutive symbols are removed from a sequence, creating a loss of information that is harder to correct due to the clustered nature of the error. As data storage and communication technologies evolve, handling burst deletions has become increasingly critical, with applications in areas such as DNA data storage (Schoeny, Sala, and Dolecek 2017), synchronizing documents (Venkataramanan, Zhang, and Ramchandran 2010), and error recovery in flash memories (Gregori et al. 2003).

1.1.1 Motivation

Error correction is fundamental to modern communication systems and data storage technologies, ensuring the reliable transmission and retrieval of information in the presence of noise and interference. Among the various types of errors encountered in these systems, burst deletions represent a particularly challenging class. A burst

deletion occurs when a contiguous sequence of symbols is removed. As an example, suppose the bit sequence $x = 1010010001$ is transmitted and $y = 1010001$ is received. Here, the error is a burst of 3 deletions, i.e., $y = 1010\cancel{010}001 = 1010001$. This type of error results in a significant loss of information and creates difficulties for error correction, especially because of the loss of synchronization. That is, the error changes the positions of the bits and so the decoder cannot be sure about any of the received bits. These challenges become even more pronounced when multiple bursts of deletions occur, as their clustered nature can severely distort the underlying data.

The motivation to address burst deletions arises from their prevalence in a wide range of applications. In DNA-based data storage, where sequences of nucleotides are used to encode digital data, burst deletions frequently occur due to sequencing errors. For instance, when sequencing reads are misaligned or certain regions of DNA are skipped, the resulting data loss can manifest as burst deletions (Field et al. 2019). Similarly, communication systems, particularly those reliant on synchronization protocols, often encounter burst deletions when a noisy channel causes the loss of multiple consecutive bits in transmitted packets (Mercier, Bhargava, and Tarokh 2010). Such errors can disrupt the synchronization of transmitted data streams, making recovery an arduous task. On the other hand, in non-volatile memory systems such as flash storage, errors often take the form of erasures, where physical wear and degradation of storage cells lead to localized block failures, resulting in the complete loss of data in specific locations.

Despite their importance, burst deletions have received comparatively less attention in the literature than other types of synchronization errors. Existing solutions tend to focus on single bursts of deletions or isolated deletions, leaving the problem of correcting multiple bursts underexplored. The challenges of burst deletion correction

are not purely theoretical but are deeply rooted in practical considerations. These include the need to balance three often competing objectives: redundancy, computational efficiency, and scalability. Achieving low redundancy is crucial for maximizing the storage or transmission efficiency of encoded data. At the same time, the computational complexity of encoding and decoding must remain manageable, particularly in systems where large-scale data is processed. Finally, scalability is essential to ensure that code constructions can handle diverse application requirements, such as sequences over larger alphabets or varying numbers of bursts.

This thesis is motivated by the need to address these challenges through extensible and modular code constructions. By designing codes that efficiently correct multiple bursts of deletions while maintaining low redundancy and manageable complexity, we aim to contribute to both the theoretical understanding and practical implementation of burst-deletion correction.

1.1.2 Background

The study of error correction plays a vital role in ensuring the reliability of modern communication systems and data storage technologies. Errors can occur during the transmission or storage of information due to noise, hardware failures, or environmental disturbances, leading to incorrect or incomplete data. Error-correcting codes are mathematical tools designed to detect and correct such errors, ensuring that the original information can be recovered even in the presence of disruptions.

Types of Errors

Errors in information systems can generally be categorized into three main types:

1. Substitution Errors

   A substitution error occurs when one symbol in a sequence is replaced by another. For example, in the binary sequence $x = (1, 0, 1, 0)$, if the third symbol is changed, the received sequence might become $y = (1, 0, 0, 0)$. Substitution errors are common in noisy communication channels and often require codes that can compare received symbols against expected patterns.

2. Insertion Errors

   An insertion error occurs when an extra, unintended symbol is added to a sequence. For instance, if $x = (1, 0, 1, 0)$, an insertion may produce $y = (1, 0, 1, 1, 0)$. Insertions disrupt the alignment of the sequence, making it difficult to match the received sequence to the original.

3. Deletion Errors

   A deletion error occurs when a symbol is removed from a sequence. For example, from $x = (1, 0, 1, 0)$, deleting the second symbol could produce $y = (1, 1, 0)$. Deletion errors are particularly challenging because they result in a loss of positional information, complicating recovery.

4. Erasure Errors

   An erasure error occurs when a symbol is removed from a sequence but its position in the sequence is known. For example, from $x = (1, 0, 1, 0)$, if the second symbol is erased, the received sequence might be represented as $y = (1, ?, 1, 0)$. Because the location of the error is known, erasure correction is often simpler than correcting deletions or substitutions.

Error-Correcting Codes

An error-correcting code (ECC) is a systematic way to encode data so that errors introduced during transmission or storage can be detected and corrected. The idea is to add redundancy to the original message, which allows the receiver to reconstruct the original information even if parts of it are lost or corrupted.

Formally, an error-correcting code maps a set of original messages (called information sequences) to a set of encoded messages. Each encoded message is called a *codeword* and the set of codewords is called a *code*. The codewords are longer than the information sequences. The difference between the length of the codewords and the information sequences is called *redundancy*. Redundancy ensures that small differences between received sequences and valid codewords, which represent errors, can be corrected. For example, a simple code might append a parity bit (an extra "0" or "1" to indicate whether the number of 1s in the sequence is even or odd) to help correct a single erasure. To illustrate, suppose we would like to transmit three bits, i.e., the set of information sequences is

$$\{000, 001, 010, 011, 100, 101, 110, 111\}$$

We use even parity by appending a parity bit such that the total number of 1s in the codeword is even. The encoding function is given in the table below:

Information Sequence	**Codeword**
000	0000
001	0011
010	0101
011	0110
100	1001
101	1010
110	1100
111	1111

So the possible codewords are:

$$C = \{0000, 0011, 0101, 0110, 1001, 1010, 1100, 1111\} \tag{1.1}$$

Suppose the received sequence is 01?1, where ? represents an erasure. Since the parity bit ensures an even number of 1s, and we observe two 1s in the received sequence, the missing bit must be 0 to maintain even parity. Thus, we correctly reconstruct the original codeword 0101, and hence the data 010.

In this example, the length of our information sequences, usually denoted k, is equal to 3. The length of our codewords, or codelength denoted by n, is 4. Hence, our redundancy here is 1 bit.

The redundancy of a code can be determined from only the code, without knowing the encoding function. Observe that the number of codewords is $|C| = 2^k$. Hence, the redundancy is $n - \log_2 |C|$ bits. If the alphabet is non-binary, and of size q, the redundancy $n - \log_q |C|$ symbols, which is equivalent to $n \log_2 q - \log_2 |C|$ bits.

Definition 1.1. The redundancy of a code C of length n over an alphabet of size q

is

$$r(C) = \log_2 \frac{q^n}{|C|}$$

bits.

Henceforth, we assume that all logarithms are to the base 2, unless otherwise stated.

We also define the error-correction capability of a code:

Definition 1.2. For a code C suppose x is transmitted and y is received, where y may have suffered a given form of errors (e.g., a given number of erasures). Then we say that C can correct this form of error if x can be recovered uniquely given y.

For instance, the code in (1.1) can correct up to 1 erasure. That is if any $x \in C$ is transmitted and y is received, possibly suffering an erasure, we can determine x from y.

Challenges in Error Correction

While error correction is a powerful tool, it comes with challenges:

1. Minimizing Redundancy for a Given Error-Correction Capability:

   Correcting errors with a large amount of redundancy is usually an easy task. But doing so will lead to high transmission or storage cost. Determining the optimal amount of redundancy and designing codes that can achieve it is often a challenging task.

2. Types of Errors:

   Sometimes errors are not of a single type, e.g., insertion, deletion, or substitution. In some applications multiple types of errors occur at the same time.

These are called edit errors. These and other types of generalized errors greatly increase the difficulty of constructing the code, but are common in practice.

3. Burst Errors:

   Errors that occur in clusters, such as burst deletions, introduce additional complexity. Achieving optimal redundancy for such errors requires more sophisticated codes capable of addressing structured errors.

1.2 Problem Statement

The central problem addressed in this thesis is the correction of multiple bursts of exactly t deletions in sequences. A burst deletion is a type of error where a contiguous block of symbols is removed, and in the case of multiple bursts, multiple such blocks are deleted from the sequence at distinct positions. As a small example, consider three bursts of four deletions occurring in a sequence x, resulting in y:

$$x = 1\,1\,0\,1\,0\,1\,0\,1\,1\,1\,0\,0\,1\,1\,0\,1\,0\,1\,1\,1\,0\,1\,0 \quad \text{(transmitted sequence)} \tag{1.2}$$

$$y = 1\,1\,\cancel{0\,1\,0\,1}\,0\,1\,1\,1\,0\,\cancel{0\,1\,1\,0}\,1\,0\,1\,1\,\cancel{1\,0\,1\,0} \quad (y \text{ with errors marked}) \tag{1.3}$$

$$y = 1\,1\,0\,1\,1\,1\,0\,1\,0\,1\,1 \quad (y \text{ as received by decoder}) \tag{1.4}$$

Our goal is to find codes such that when x belongs to these codes, we can recover it from y. These errors pose significant challenges for error correction due to the structured and clustered nature of the deletions, which disrupt the alignment and recovery of the original data.

The goal of this research is to design efficient error-correcting codes that:

- Correct two or more bursts, each consisting of exactly t deletions, in both binary and q-ary sequences.
- Minimize the redundancy required to protect the original data.
- Ensure scalability to larger alphabets, a larger number of bursts, and longer bursts, while maintaining manageable computational complexity.

By tackling these problems, this work aims to advance the understanding and practical application of error correction in scenarios involving burst deletions.

1.3 Research Contributions

While existing codes, such as the Varshamov-Tenengolts (VT) code and its extensions, effectively address single deletions and some random deletions, their applicability to burst deletions—especially multiple bursts—remains limited. Recent works, such as Ye et al.'s construction (Ye, Yu, and Elishco 2024), have focused on two bursts of deletions with minimal redundancy, but they rely on specially tailored erasure-correcting codes that lack extensibility to larger numbers of bursts. In contrast, the constructions presented in this thesis not only achieve competitive redundancy but also provide extensive and modular frameworks that enable extensibility to more complex error scenarios, such as correcting three or more bursts of deletions.

This thesis presents novel code constructions for correcting multiple bursts of exactly t deletions in sequences, with significant improvements in redundancy and computational efficiency compared to existing methods. The key contributions of this research are as follows:

1. Code for Correcting Two Bursts of t Deletions in Binary Sequences:
   - This work constructs a code capable of correcting two bursts of t deletions in binary sequences.
   - The proposed code achieves a redundancy of $6.4 \log n + 10 \log \log n + O(1)$ demonstrating an efficient balance between error-correction capability and storage overhead.
2. Code for Correcting Two Bursts of t Deletions in q-ary Sequences:
   - The binary construction is extended to q-ary sequences, resulting in a code that corrects two bursts of t deletions with redundancy of $7.4 \log n + 10 \log \log n + 3 \log q + O(1)$.
   - This generalization provides scalability to sequences over larger alphabets, making the code applicable to a wider range of practical scenarios, such as DNA data storage.
   - Compared to existing approaches, this code has versatile generalizability, which allows it to be easily extended to more bursts of exactly t deletions.
3. Code for Correcting Three Bursts of t Deletions:
   - We proposed a novel code construction for correcting three bursts of t deletions.
   - The code achieves a redundancy of at most $15 \log n + o(\log n)$ with an encoding and decoding complexity of $O(n^7)$.
   - While the redundancy is higher than directly using syndrome compression, the proposed construction offers a significantly lower decoding complexity $O(n^7)$ vs. $O(n^7 q^t)$, making it computationally feasible for large t.

Overall, the contributions of this thesis advance the field of error-correcting codes by addressing the challenges of burst deletions in both theory and application. These results lay a foundation for further improvements in correcting clustered errors in communication systems, data storage, and related fields.

1.4 Thesis Organization

This thesis is organized as follows. In Chapter 2, we provide a brief introduction to error correcting codes and some relevant prior work. In Chapter 3, we present three codes that can correct i) two bursts of t deletions in binary, ii) two bursts of t deletions in q-ary, and iii) three bursts of exactly t deletions in q-ary sequences. For each code, we provide the code construction and find the redundancy and time complexity. In Chapter 4, we discuss the pros and cons of our codes compared to prior work, and provide practical applications and future directions for our research. In Chapter 5, we summarize the main achievements of our research. Lastly, in Chapter 6, we provide a summary of the entire thesis.

Chapter 2

Review of Literature

This chapter provides an overview of the foundational work and recent advancements in error-correcting codes relevant to burst deletion errors. We begin by introducing key definitions and concepts in Section 2.1, followed by a discussion of notable prior work on correcting single, double, and burst deletion errors in Section 2.2. Finally, we highlight the limitations and challenges of these existing methods in Section 2.3, underscoring the gaps that this thesis seeks to address.

2.1 Preliminaries

In this section, we introduce key definitions, notation, and representations that form the foundation for the constructions proposed in this thesis. These concepts will be used throughout the subsequent chapters to describe, analyze, and evaluate the proposed error-correcting codes for burst deletions.

A sequence is an ordered list of symbols, typically binary (0s and 1s) or from a larger q-ary alphabet. Errors such as deletions, insertions, and substitutions disrupt these sequences, potentially resulting in data loss or corruption.

Let $\mathbb{F}_2^n$ denote all binary sequences of length n. A run of length P of a sequence x is a substring of x such that $x_i = x_{i+1} = \cdots = x_{i+P-1}$, in which $x_{i-1} \neq x_i$ if $i > 1$ and

$x_{i+P-1} \neq x_{i+P}$ if $i + P - 1 < n$. We use $\mu(x)$ to denote the length of the longest run in x.

A *burst* of length b is defined as the deletion or insertion of t consecutive symbols from a sequence. Formally, if $x = (x_1, x_2, \ldots, x_n)$ is a sequence, a burst of deletions of length b results in the subsequence $(x_1, \ldots, x_{i-1}, x_{i+b}, \ldots, x_n)$ for some $1 \leq i \leq n - b + 1$.

We refer to a two-burst-t-deletion error when exactly two bursts of consecutive deletions of length t have occurred, i.e., from x, we obtain a subsequence

$$(x_1, ..., x_a, x_{a+t+1}, ..., x_b, x_{b+t+1}, ..., x_n) \in \mathbb{F}_2^{n-2t}$$

The two-burst-t-deletion ball of a sequence $x \in \mathbb{F}_2^n$, is denoted by $D_t^2(x)$, and is defined to be the set of subsequences of x of length $n - 2t$ obtained by the deletion of two bursts of size t.

A two-burst-t-deletion-correcting code C is a set of sequences, each called a codeword, in $\mathbb{F}_2^n$ such that there are no two codewords in C where two deletion bursts of size t result in the same word of length $n-2t$. That is, for every $x, y \in C, D_t^2(x) \cap D_t^2(y) = \emptyset$.

In this paper, we let t be a fixed integer that divides n. We denote sequences with lower-case letters and the corresponding $t \times \frac{n}{t}$ matrices with upper-case letters. For a vector $x = (x_1, x_2, ..., x_n)$, we define the following $t \times \frac{n}{t}$ matrix:

$$X = \begin{bmatrix} x_1 & x_{t+1} & \cdots & x_{n-t+1} \\ x_2 & x_{t+2} & \cdots & x_{n-t+2} \\ \vdots & \vdots & \ddots & \vdots \\ x_t & x_{2t} & \cdots & x_n \end{bmatrix}$$

and for $1 \leq i \leq n$, we denote $X_{(i)}$ as the i-th row of the matrix X, and X_j as the j-th column of X. This representation allows errors to manifest as disruptions in specific rows or columns, simplifying the correction process.

We provide the following example to demonstrate the above definitions.

Example 2.1. Let $x = 0100100011011101$. There are 10 runs in x, and the longest run is of length 3. The matrix X, for $t = 4$, is formed as follows:

$$X = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

Let 1001 and 0110 be the two substrings that are removed from x. Let $y = 00011101$ be the received string, obtained from x by two bursts of t deletions. Then Y is formed as follows:

$$Y = \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}$$

This representation helps isolate the affected areas and enables targeted error correction.

We will use a small example to show why matrix formation is useful in burst deletions.

Example 2.2. Consider a burst of 4 deletions occurs in the sequence x, resulting in y:

$$x = 1\,1\,0\,1\,0\,1\,0\,1\,1\,1\,0\,0\,1\,1\,0\,1 \quad \text{(transmitted sequence)} \tag{2.1}$$

$$y = 1\,1\,0\,1\,0\,\cancel{1\,0\,1\,1}\,1\,0\,0\,1\,1\,0\,1 \quad (y \text{ with errors marked}) \tag{2.2}$$

$$y = 1\,1\,0\,1\,0\,1\,0\,0\,1\,1\,0\,1 \quad (y \text{ as received by decoder}) \tag{2.3}$$

If we write this in matrix form, then we have:

$$X = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix} \quad \Rightarrow \quad Y = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

Notice that exactly one bit is removed from each row of the matrix. Specifically, in the first row, the third bit 1 is deleted; in the second row, the second bit 1 is deleted; in the third row, the second bit 0 is deleted; and in the last row, the second bit 1 is deleted. By treating each row as an independent sequence, the problem of a single burst of four deletions is effectively reduced to four separate single-deletion problems. This key idea will serve as a foundational approach throughout the paper.

2.2 Relevant Prior Work

This section provides an overview of prior work related to error-correcting codes for deletions and burst deletions. We begin with foundational codes like the Varshamov-Tenengolts (VT) code for single deletions, then discuss progress in correcting two deletions, and conclude with burst-deletion correcting codes. These works form the basis for the constructions proposed in this thesis.

The Varshamov-Tenengolts (VT) code was initially designed to correct asymmetric errors (Varshamov and Tenengolts 1965), such as substitution errors that occur more frequently in one direction (e.g., flipping 0 to 1 but not the reverse). Later, Levenshtein demonstrated that the VT code could also correct a single insertion or deletion error, making it one of the first error-correcting codes capable of addressing this type of error. This versatility arises from the arithmetic property (known as the VT syndrome), which allows the code to efficiently identify and correct such errors. The VT code is defined as follows:

Definition 2.3 (Varshamov and Tenengolts 1965)**.** The *Varshamov-Tenengolts (VT) code* of length n, denoted as $VT_a(n)$, where a is a fixed integer, is defined as the set of binary codewords $x = (x_1, x_2, \ldots, x_n) \in \{0,1\}^n$ that satisfy the following congruence condition:

$$\sum_{i=1}^{n} i \cdot x_i \equiv a \pmod{(n+1)}$$

In other words, the VT code consists of all binary sequences of length n for which the weighted sum of the bits (where the weight of the bit in position i is i) is congruent to a modulo $n+1$. We call a the VT syndrome.

Correcting two deletion errors is significantly more challenging than correcting a

single deletion because the relationship between the remaining symbols becomes more ambiguous. While the VT code efficiently corrects a single insertion or deletion, it took nearly 60 years for a code capable of correcting two deletions with redundancy close to the existential bound to be proposed. For 2-deletion codes, the existential bound ensures the existence of codes with redundancy $4\log n + o(\log n)$, derived from combinatorial arguments. This bound was matched by the syndrome compression codes and Guruswami and Håstad's explicit construction in 2021 (Guruswami and Håstad 2021).

Based on Guruswami and Håstad's work, Liu, Tjuawinata, and Xing (2024) proposed an efficient q-ary two-deletion correcting code that is defined as follows:

Theorem 2.4. *Let $C_0 \subseteq \Sigma_2^n$ be the binary two-deletion correcting code constructed by Guruswami and Håstad (2021). Then for any positive integer $q > 2$, there exists an explicit and efficiently encodable 2 deletion correcting code $C_1 \subseteq \Sigma_q^n$ with redundancy $5\log n + 10\log\log n + 3\log q + O(1)$.*

Burst errors, where multiple consecutive symbols are inserted or deleted, are common in storage systems and DNA sequence analysis. Schoeny et al. (C. Schoeny, Wachter-Zeh, et al. 2017) (C. Schoeny, F. Sala, and L. Dolecek 2017) developed a code that can efficiently correct a single burst of deletions with redundancy proportional to $\log n$, making it suitable for long sequences.

In this work, we also use a extensive k-deletion correcting code using syndrome compression in the case where the original sequence x contains more than 2 bursts of deletions. This k-deletion correcting code using syndrome compression is defined as follows:

Lemma 2.5 (Syndrome Compression for k Deletions (Sima, Gabrys, and Bruck

2020a)). *Let $n \geq 3$ and $k \geq 1$ be integers. There exists a function*

$$S_k : \{0,1\}^n \longrightarrow \{0,1\}^{4k \log n + o(\log n)},$$

computable in $O(n)$ time, with the following property: for any string $x \in \{0,1\}^n$ and any string y obtained from x by up to k deletions, one can uniquely and efficiently recover x given y and $S_k(x)$.

The paper by Dumer Dumer 1995 provides infinite families of linear codes of designed distance $\delta = 4$, 5, and 6. Specifically, let I_4, I_5, I_6 be the set of code zeros used to define the cyclic code, P^{m-1} be the projective space of dimension, and X_5, X_6 be the cubic variety in the affine space F^{m-1} for $\delta = 5, 6$, the following results are presented:

Lemma 2.6 (Corollary 5 from Dumer 1995). *For $m = 2, 4, \ldots,$ there exists a code $V_q^m(I_4, P^{m-1})$ with the following parameters:*

$$n = \frac{q^m - 1}{q - 1}, \quad r(q, n, 4) = \frac{3m}{2}, \quad \delta = 4,$$

where $r(q, n, 4)$ denotes the redundancy of the code.

Lemma 2.7 (Corollary 6 from Dumer 1995). *For $m = 3, 5, \ldots,$ there exists a code $V_q^m(I_5, X_5)$ with the following parameters:*

$$n = q^{\lfloor 5(m-1)/6 \rfloor}, \quad r(q, n, 5) \leq 2m, \quad \delta = 5,$$

where $r(q, n, 5)$ denotes the redundancy of the code.

Lemma 2.8 (Corollary 6 from Dumer 1995). *For $m = 4, 6, \ldots,$ there exists a code*

$V_q^m(I_6, X_6)$ *with the following parameters:*

$$n = q^{\lfloor 5(m-1)/6 \rfloor}, \quad r(q, n, 6) \leq 2.5m, \quad \delta = 6,$$

where $r(q, n, 6)$ *denotes the redundancy of the code.*

Correcting two bursts of deletions is a challenging problem because the deletions create ambiguities in the sequence that are difficult to resolve. The results presented by Ye, Yu, and Elishco (2024) address this by designing explicit constructions that encode additional information about the sequence in a structured manner, enabling efficient recovery.

In Ye et al's paper (Ye, Yu, and Elishco 2024), the authors introduce a construction for non-binary codes with improved redundancy, enabling the correction of up to two disjoint bursts of t deletions with redundancy $5 \log n + O(\log \log n)$. The construction is as follows:

Theorem 2.9 (Correcting Two Bursts of Exactly t Deletions (Ye, Yu, and Elishco 2024))**.** *Suppose* $q > 2$. *Let* $N_1, N_2, Q, f_0(x), f_1(x), h^{(0)}(x)$ *and* $h^{(1)}(x)$ *be defined as in [Ye, Yu, and Elishco 2024]. For all* $0 \leq a < 2^{4 \log n + 10 \log \log n + O(1)}$, $0 \leq b_0 < 2N_1$, $0 \leq b_1 < Q$, *and* $0 \leq c_0, c_1 < N_2$, *define the code* C_2 *as*

$$C_2 \triangleq \left\{ x \in \Sigma_q^n : \begin{array}{l} x \text{ is } d\text{-regular}, \\ \eta(u(x)) = a, \\ f_s(x) = b_s \text{ for } s \in \{0, 1\}, \\ h^{(0)}(x) = c_0,\ h^{(1)}(x) = c_1 \end{array} \right\}.$$

Then C_2 *is a code correcting two bursts of exactly* b *deletions.*

Ye, Yu, and Elishco (2024) also provide an efficient code correcting two bursts of two erasures. Let $N \geq 2$ be an integer. Suppose $x \in [0, N-1]^n$, where $n > 4$. Let ? denote an unknown symbol. If $y \in ([0, N-1] \cup \{?\})^n$ is such that $y_i = y_{i+1} =?$, $y_j = y_{j+1} =?$, and $y_k = x_k$ for all $k \notin \{i, i+1, j, j+1\}$, we say that y is obtained from x by two bursts of erasures (of length two). As a small example, consider two bursts of two erasures occurring in a sequence x, resulting in y:

$$x = 1\,1\,0\,1\,0\,1\,0\,1\,1\,1\,0\,0\,1\,1\,0\,1\,0\,1\,1\,1\,0\,1\,0 \quad \text{(transmitted sequence)} \tag{2.4}$$

$$y = 1\,1\,0\,1\,0\,\not{1}\not{<}\,0\,1\,1\,1\,0\,0\,\not{1}\not{<}\,0\,1\,0\,1\,1\,1\,0\,1\,0 \quad (y \text{ with errors marked}) \tag{2.5}$$

$$y = 1\,1\,0\,1\,?\,?\,0\,1\,1\,1\,0\,0\,?\,?\,0\,1\,0\,1\,1\,1\,0\,1\,0 \quad (y \text{ as received by decoder}) \tag{2.6}$$

Assume $j \geq i+2$, for a sequence x over the alphabet $[0, N-1]$, denote $A(x) = A(x, |x|, 2)$ and $\mathrm{Syn}(x) \triangleq \sum_{i=1}^{n} i x_i$.

Lemma 2.10 (Lemma II.7)**.** *For any $0 \leq a_1, a_2 < 2N$ and $0 \leq b < nN^2$, define C to be the set of all sequences $x \in [0, N-1]^n$ that satisfy:*

(C1) $\sum_{j=1}^{\lceil n/2 \rceil} A(x)_{1,j} \equiv a_1 \pmod{2N}$, $\sum_{j=1}^{\lceil n/2 \rceil} A(x)_{2,j} \equiv a_2 \pmod{2N}$;

(C2) $W(x) \equiv b \pmod{nN^2}$, *where* $W(x) = Syn(A(x)_1) + (2N-1) \cdot Syn(A(x)_2)$.

In particular, there's a function $\rho : [0, N-1]^n \to \Sigma_2^{\log n + 4\log N + 2}$, efficiently computable, such that for any $x \in [0, N-1]^n$, given $\rho(x)$, we can efficiently and uniquely recover x from y, where y is any sequence obtained from x by two bursts of two erasures.

2.3 Limitations of Prior Work

Despite significant advancements in error-correcting codes, the problem of correcting multiple bursts of deletions has remained underexplored. Most prior work has focused on single or double deletion errors, leaving the challenges of burst deletion correction largely unaddressed. Among the limited research, the work by Ye, Yu, and Elishco (2024) provides an explicit construction for codes capable of correcting two bursts of exactly t deletions. However, even this approach faces certain limitations.

1. Limited Applicability to Multi-Burst Errors: While Ye, Yu, and Elishco (2024) achieve efficient redundancy of $5\log n + O(\log\log n)$ for two bursts of exactly t deletions, their construction is not extensive. This lack of extensibility makes it challenging to extend their method to scenarios involving more than two bursts of deletions. For applications requiring robustness against multiple bursts, this is a significant limitation.

2. Redundancy vs. Extensibility Trade-Off: Comparing the code correcting two bursts of exactly t deletions presented in this thesis with the construction in Ye, Yu, and Elishco 2024, it is true that the redundancy of our code is higher. However, the modularity of our approach allows for straightforward extensions to handle additional bursts, making it a more flexible solution for cases with higher deletion complexity.

Symdrome Compression also has limitation for large t. Syndrome compression (Sima, Gabrys, and Bruck 2020a) is an alternative technique that can address three bursts of exactly t deletions with lower redundancy than our code. However, this method comes at the cost of significantly higher decoding complexity, particularly when t

is large. In contrast, our construction achieves a more efficient decoding process, making it better suited for applications requiring low time complexity.

In summary, while prior work has made strides in addressing specific cases of burst deletions, significant gaps remain:

- Extending solutions to multiple bursts in a extensive and modular fashion remains challenging.
- Balancing redundancy and computational efficiency, especially for larger t, is still an open problem.
- Existing methods often trade extensibility or efficiency for optimal redundancy, limiting their practical applicability in broader contexts.

The limitations of existing approaches highlight the need for codes that strike a balance between redundancy, extensibility, and computational efficiency. These challenges motivate the constructions presented in this thesis, which aim to address the gaps in the literature while offering a extensive and flexible framework for correcting multiple bursts of deletions.

Chapter 3

Code Construction

This chapter presents the construction of three codes designed to correct bursts of deletions in binary and q-ary sequences. Each construction builds upon foundational techniques, such as Guruswami and Håstad's two-deletion correcting code (Guruswami and Håstad 2021) and syndrome compression (Sima, Gabrys, and Bruck 2020a), and adapts them to address the unique challenges posed by burst deletions. By leveraging matrix representations and efficient error localization and correction techniques, we achieve extensive and scalable solutions.

The chapter is divided into three sections:

- Code for Two Bursts of Exactly t Deletions in Binary Sequences: This section introduces a binary code that combines Guruswami and Håstad's code for error localization with a distance 5 code for error correction.
- Code for Two Bursts of Exactly t Deletions in q-ary Sequences: Building on the binary construction, this section extends the methodology to q-ary sequences using Liu et al.'s (Liu, Tjuawinata, and Xing 2024)two-deletion correcting code.
- Code for Three Bursts of Exactly t Deletions: The final section presents a novel approach to handle three bursts of t deletions, utilizing syndrome compression and adaptations of Dumer's code to balance redundancy and computational efficiency.

Through detailed mathematical constructions and proofs, this chapter establishes the correctness, redundancy, and computational feasibility of the proposed codes.

3.1 Code Correcting Two Bursts of Exactly t Deletions in Binary Sequences

This section presents the construction of a code that can correct two bursts of t deletions in binary sequences. The construction combines Guruswami and Håstad's two-deletion correcting code (Guruswami and Håstad 2021) with a distance-5 code (Dumer 1995) to address errors localized in specific rows of a matrix representation.

3.1.1 Overview

Given a binary sequence x of length n, the goal is to correct two bursts of t deletions. We achieve this by:

- Error Localization: Use the first row of a matrix representation of x to identify approximate positions of the deletions.

- Error Correction: Correct errors in the remaining rows using an efficient erasure-correcting code.

This approach ensures low redundancy while maintaining computational efficiency.

3.1.2 Definitions and Lemmas Used

In this subsection, we describe all definitions and lemmas that we will use in our construction.

Definition 3.1 (regularityGuruswami and Håstad 2021)**.** A sequence $x \in \Sigma_2^n$ is a regular sequence of length n if any substring of x of length at least $d \log n$ contains both 00 and 11.

Here d is an absolute fixed constant, so we use $d = 7$ in the rest of the paper.

Lemma 3.2 (Guruswami and Håstad's Two Deletion Correcting Code (Guruswami and Håstad 2021))**.** *Suppose $d \geq 7$, there exists a function*

$$GH : \Sigma_2^n \to \Sigma_2^{4\log n + 10 \log\log n + O(1)}$$

computable in linear time, such that for any regular sequence $x \in \Sigma_2^n$, given $GH(x)$ and any y obtained by deleting two bits from x, one can uniquely and efficiently recover x.

If the first row $X_{(1)}$ is regular, we can use the above code to correct the error in $X_{(1)}$. This will tell us the approximate positions of the two deletions in $X_{(1)}$. Let $P = 7 \log n$, then the longest run in $X_{(1)}$ has length at most P since it is 7-regular. For each of the remaining rows from $X_{(2)}$ to $X_{(t)}$, we divide every row into $\frac{n}{Pt}$ blocks, where each block is of length P. The number of error runs in $X_{(1)}$ is either 1 or 2, and one run in $X_{(1)}$ can affect at most 2 blocks in the remaining rows. Since we do not know whether the left block or the right block contains the error, this problem turns into a 4-erasure problem.

To correct these erasures, we use a code with designed distance 5, which can correct up to 4 erasures. Specifically, we use the redundancy function associated with the code in Lemma 2.7. Define the **syndrome function** $D_5(x)$ to be the function that maps the information part of a sequence to its redundancy symbols in the code $V_q^m(I_5, X_5)$ from Lemma 2.7. These redundancy symbols serve as compressed information that allows us to recover erased values.

Lemma 3.3 (Sima, Raviv, and Bruck 2019)**.** *For any integer $n \geq 3$, there exists a function $S : \Sigma_2^n \to \Sigma_2^{7\log n + O(1)}$, computable in linear time, such that for any $x \in \Sigma_2^n$, given $S(x)$ and any $y \in B_1^2(x)$, one can uniquely and efficiently recover x.*

Since every block may contain 0, 1, or 2 deletions, we apply the above code to correct each block. Here we do not use Guruswami and Håstad's two-deletion correcting code because we do not want to restrict every block to be regular. Now we can give a formal construction of our code that corrects two bursts of t deletions.

3.1.3 Construction

In this subsection, we will provide a detailed set of steps to construct our code, a mathematical construction, and a proof of the code's correctness and redundancy.

To construct our code that corrects two bursts of t deletions, we will follow the following steps:

1. Matrix Representation: The binary sequence x of length n is represented as a $t \times \frac{n}{t}$ matrix X, where each row corresponds to a subsequence of x. Each row $X_{(i)}$ is further divided into blocks of length $P = 7\log n$.

2. Regularity Condition: To enable effective correction in the first row, we require

it to be "regular," meaning that any sufficiently long substring contains both "00" and "11" patterns. This ensures the error localization process is robust.

3. Error Localization in the First Row: We apply Guruswami and Håstad's Code, which corrects two deletions in a regular sequence, to the first row $X_{(1)}$. This identifies approximate positions of the two deletions.

4. Error Correction in Remaining Rows: Using the approximate deletion positions in $X_{(1)}$, we identify at most 4 affected blocks across the remaining rows. These blocks are treated as containing erasures. Each block is corrected using a code with distance 5 (Dumer's Code), which efficiently handles up to 4 erasures.

The overall code is defined as follows:

Construction 3.4. To correctly recover the erased blocks, we need a code with designed distance 5, which can correct up to 4 erasures. We use the redundancy function associated with the code in Lemma 2.7.

Encoding Process Denote $X_{i,j}$ as the j-th block of row i. Define

$$\overline{S}(X_c) = S(X_{2,c})S(X_{3,c})\dots S(X_{t,c})$$

as the concatenation of the syndromes of each block in column c that are generated using the above code. Note that as the length of each $X_{i,c}$ is $O(\log n)$ bits, by Lemma 3.3, the length of $S(X_{i,c})$ is $O(\log\log n)$ bits. We encode each $\overline{S}(X_c)$ as a symbol in the codeword of Dumer's code with designed distance 5 and alphabet size $q_{D_5} = 2^{O(t\log\log n)}$. To correct erasures, define:

$$f(x) = D_5(\overline{S}(X_1), \overline{S}(X_2), \dots, \overline{S}(X_{\frac{n}{Pt}})).$$

Choice of Parameters We select $P = 7\log n$ to ensure that the longest run in $X_{(1)}$ has length at most P, since $X_{(1)}$ is 7-regular. The matrix representation has $\frac{n}{Pt}$ blocks in each row.

To apply Dumer's code, we determine appropriate values of q and m. The alphabet size of Dumer's code is $q_{D_5} = 2^{O(t\log\log n)}$. The parameter m is selected as the smallest odd number satisfying:

$$q^{\lfloor 5(m-1)/6 \rfloor} - 2.5m \geq \frac{n}{Pt}.$$

which leads to $m \simeq \frac{6}{5}\log_q \frac{n}{Pt}$.

If the smallest m satisfying this condition yields a longer code than needed, we apply **shortening** by fixing Z positions to predetermined values before computing the syndrome.

Definition of the Code Suppose $n \geq 2t$ and $t \geq 1$. For $0 \leq a \leq 2^{4\log n + 10\log\log n + O(1)}$ and $0 \leq b \leq 2^{7\log n/3}$, define the two-burst-$t$-deletion correcting code C_1 as:

$$C_1 \triangleq \left\{ \begin{array}{c} X_{(1)} \text{ is regular}, \\ x \in \Sigma_2^n : GH(X_{(1)}) = a, \\ f(x) = b \end{array} \right\}.$$

The total redundancy of this code is computed as follows:

- Redundancy for Error Localization: Using Guruswami and Håstad's Code adds a redundancy of $4\log n + 10\log\log n + O(1)$.
- Redundancy for Erasure Correction: According to Lemma 2.7, the total redun-

dancy for correcting the remaining rows is at most $2m \sim 2 \times \frac{6}{5} \log n = 2.4 \log n$.

Summing these, the overall redundancy of the code is at most $6.4 \log n + 10 \log \log n + O(1)$. This redundancy is higher than that of the existing code in Ye, Yu, and Elishco (2024), but our approach has the advantage of being extendable to cases with more than two bursts.

Below we have a lemma and its proof for the correctness and redundancy of our code.

Lemma 3.5. *The code C_1 as defined above has redundancy at most $6.4 \log n + 10 \log \log n + O(1)$ and can correct up to two deletions in binary sequences.*

Proof. The first part of our code is to locate the approximate locations of the two missing bits in the first row of the matrix. This is achieved by applying Guruswami and Håstad's code as it can help us to correct two deletions in a binary sequence. This adds $4 \log n + 10 \log \log n + O(1)$ to our redundancy.

All that remains is to calculate the redundancy needed to correct the remaining rows of the matrix. As a recap, in this part we treat the 4 potential error columns as a four erasure problem and use Dumer's code with designed distance 5 to correct the erasures. The size of the syndrome in each block is $O(\log \log n)$ bits, and each concatenation of syndromes contains $t - 1$ values, so the total size of a single concatenation is $O((t-1) \log \log n) = O(\log \log n)$ bits. This is also the size of q in Dumer's code with designed distance 5. According to Lemma 2.7, the redundancy of the remaining rows is $2m \sim 2.4 \log_q \frac{n}{Pt}$ symbols $\leq 2.4 \log n$ bits.

Summing them up, the total redundancy of our code that corrects two-burst-t-deletion is at most $4 \log n + 10 \log \log n + O_d(1) + 2.4 \log n = 6.4 \log n + 10 \log \log n + O(1)$.

The error-correction capability follows from our discussion preceding the code con-

struction. □

3.2 Code Correcting Two Bursts of Exactly t Deletions in q-ary Sequences

Recall that in the previous subsection we break the problem of correcting 2 bursts of t deletions into two parts. We first use Lemma 3.2, Guruswami and Håstad's two deletion correcting code, to find the approximate positions of the two missing bits, then use an erasure correcting code, Dumer's code with designed distance 5, to correctly recover the message. Note that extending this to q-ary sequences only requires us to find a extensive q-ary two deletion correcting code to find the error positions. Once we find the approximate positions of the deletions, we can still use Dumer's code to recover the original message.

3.2.1 Overview

Given a q-ary sequence x of length n, the goal is to correct two bursts of t deletions. We achieve this by:

- Error Localization: Use the first row of a matrix representation of x to identify approximate positions of the deletions.
- Error Correction: Correct errors in the remaining rows using an efficient erasure-correcting code.

This approach ensures low redundancy while maintaining computational efficiency.

3.2.2 Construction

Let $P = \log n$ and restrict the longest run in $X_{(1)}$ to have length at most P. Let $L(x)$ be the code defined in Theorem 2.3. Since $L(x)$ is capable of correcting two deletions in a q-ary sequence, we can use this code to correct the errors in $X_{(1)}$. This will tell us the approximate positions of the two deletions in $X_{(1)}$. Similar to Section 3.1, since there are only two deletions in $X_{(1)}$, we will find at most 2 runs that contain deletions.

We can again divide every other row into $\frac{n}{Pt}$ blocks, where each block is of length P. Similar to the previous section, to correctly recover the information in these rows, we can view this problem as a 4-erasure problem. We can apply $L(x)$ again to every block to generate syndromes of each block, and again concatenate all syndromes of the blocks from the same column together. Below is the formal construction of our code that corrects 2 bursts of t deletions in a q-ary sequence.

To construct our code that corrects two bursts of t deletions, we will follow the following steps:

1. Matrix Representation: The q-ary sequence x of length n is represented as a $t \times \frac{n}{t}$ matrix X, where each row corresponds to a subsequence of x. Each row $X_{(i)}$ is further divided into blocks of length $P = \log n$.

2. Restriction in Run Length: To enable effective correction using Dumer's Code, we restrict the run length limit in the first row $X_{(1)}$, specifically the length of the longest run in the first row $X_{(1)}$ is at most $\log n$.

3. Error Localization in the First Row: We apply Liu et al.'s Code, which corrects two deletions in a q-ary sequence, to the first row $X_{(1)}$. This identifies

approximate positions of the two deletions.

4. Error Correction in Remaining Rows: Using the approximate deletion positions in $X_{(1)}$, we identify at most 4 affected blocks across the remaining rows. These blocks are treated as containing erasures. Each block is corrected using a code with distance 5 (Dumer's Code), which efficiently handles up to 4 erasures.

The overall code is defined as follows:

Construction 3.6. Denote by $X_{i,j}$ the j-th block of row i. Define

$$\overline{L}(X_c) = L(X_{2,c})L(X_{3,c})\dots L(X_{t,c})$$

as the concatenation of the syndromes of each block in column c that are generated using the code in Theorem 2.3. These syndromes serve as compressed information that allows us to correct potential erasures in each block.

Encoding Process To recover the erased values, we use a code with designed distance 5, which can correct up to 4 erasures. We encode each $\overline{L}(X_c)$ as a symbol in the codeword of Dumer's code with designed distance 5 and alphabet size $q_{D_5} = 2^{O(t \log\log n)}$. To correct erasures, define:

$$f_q(x) = D_5(\overline{L}(X_1), \overline{L}(X_2), \dots, \overline{L}(X_{\frac{n}{Pt}})).$$

Erasure Correction with Dumer's Code As in the binary case, we use the redundancy function associated with the code in Lemma 2.7.

To apply Dumer's code, we determine appropriate values of q and m similarly to the

binary case, ensuring that:

$$q^{\lfloor 5(m-1)/6 \rfloor} - 2.5m \geq \frac{n}{Pt}.$$

which leads to $m \simeq \frac{6}{5} \log_q \frac{n}{Pt}$.

Similarly, if the smallest odd m satisfying this condition yields a longer code than needed, we apply **shortening** by fixing Z positions to predetermined values before computing the syndrome.

Definition of the Code Suppose $q \geq 2$, $n \geq 2t$, and $t \geq 1$. For $0 \leq a \leq 2^{5\log n + 10\log\log n + 3\log q + O(1)}$ and $0 \leq b \leq 2^{7\log n/3}$, define the code C_2 that corrects two bursts of exactly t deletions in q-ary sequences as:

$$C_2 \triangleq \left\{ \begin{array}{c} \mu(X_{(1)}) \leq \log n, \\ x \in \Sigma_q^n : L(X_{(1)}) = a, \\ f_q(x) = b \end{array} \right\}$$

where $\mu(X_{(1)})$ is the length of the longest run in the first row.

The total redundancy of this code is computed as follows:

- Redundancy for Error Localization: Using Liu et al.'s Code adds a redundancy of $5\log n + 10\log\log n + 3\log q + O(1)$.
- Redundancy for Erasure Correction: According to Lemma 2.7, the total redundancy for correcting the remaining rows is at most $2.4\log n$.

Summing these, the overall redundancy of the code is at most $7.4\log n + 10\log\log n + 3\log q + O(1)$. This redundancy is higher than that of the existing code in Ye, Yu, and Elishco 2024, but our approach has the advantage of being extendable to cases with more than two bursts.

Below we have a lemma and its proof for the correctness and redundancy of our code.

Lemma 3.7. *The code C_2 as defined above has redundancy at most $7.4\log n + 10\log\log n + 3\log q + O(1)$ and can correct up to two bursts of t deletions in q-ary sequences.*

Proof. Similar to our proof in Lemma 3.5, we first use the code in Theorem 2.3 and this adds $5\log n + 10\log\log n + 3\log q + O(1)$ to our redundancy. In the remaining rows, Dumer's code with designed distance 5 adds at most $2.4\log n$ to our redundancy according to Lemma 2.7. The total redundancy of C_2 that corrects two-burst-t-deletion is at most $7.4\log n + 10\log\log n + 3\log q + O(1)$. The error-correction capability follows from our discussion preceding the code construction. □

3.3 Code Correcting Three Bursts of Exactly t Deletions

In this section, we extend the idea from the previous section to construct a code correcting three bursts of t deletions with redundancy at most $15\log n + o(\log n)$. We prove that although this redundancy is slightly higher than directly using syndrome compression to correct 3 bursts of t deletions, the computational complexity of our code is much lower when t is large. Our approach is to first use syndrome compression

in the first row $X_{(1)}$ to find the approximate positions of the missing bits, then correct these intervals to recover the original sequence.

3.3.1 Overview

Given a q-ary sequence x of length n, the goal is to correct three bursts of t deletions. We achieve this by:

- Error Localization: Use the first row of a matrix representation of x to identify approximate positions of the deletions.
- Error Correction: Correct errors in the remaining rows using an efficient erasure-correcting code.

This approach ensures low redundancy while maintaining computational efficiency.

3.3.2 Extending From Two Bursts to Three Bursts

Recall that in the previous section we first locate positions of deletions in short intervals, then correct errors in these intervals. We will use a similar approach here. Now in $X_{(1)}$ there are three deletions, so we will use the extensive k deletion correcting code in Lemma 2.4 to locate those deletions. In this case, $k = 3$, so this code needs a redundancy of $12 \log n + o(\log n)$ to correct three deletions.

Let $P = \log n$. We restrict $X_{(1)}$ to be P-run length limited. We divide each following row into blocks of length P. Since there are 3 deletions in $X_{(1)}$, we will either have 1) three deletions in three distinct runs, 2) two deletions in one run and one deletion in another run, or 3) three deletions in the same run. Since each run can affect at

most two blocks in the following rows, at most 6 blocks can be affected in each of the rows. This turns the problem into a 6 erasure problem.

Using a Reed-Solomon code to correct these 6 erasures results in a relatively high redundancy, and in [Dumer 1995] Dumer only provides codes with design distances 4, 5, and 6 which cannot correct 6 erasures. To correct 6 erasures, we need a code with minimum distance 7. Inspired by Ye, Yu, and Elishco 2024, we prove that Dumer's code can be used in this scenario because instead of a normal 6 erasure problem, our problem is more similar to three bursts (of length two) of erasures. Each burst of length two of erasures is generated by the approximate position of the missing bit in $X_{(1)}$.

Lemma 3.8. *Let $a \in \Sigma_q^n$ be a sequence over a q-ary alphabet, and let $b \in \Sigma_q^{n-6}$ be obtained from a by three bursts of 2 erasures. Define A as a $2 \times \frac{n}{2}$ matrix representation of a, where:*

- *The first row $A_{(1)}$ contains all odd-indexed bits of a, and*
- *The second row $A_{(2)}$ contains all even-indexed bits of a.*

Let B be the corresponding matrix representation of b, such that:

- *The first row $B_{(1)}$ corresponds to the sequence obtained from $A_{(1)}$ after erasing bits affected by the three bursts of erasures in a, and*
- *The second row $B_{(2)}$ corresponds to the sequence obtained from $A_{(2)}$ after erasing bits affected by the three bursts of erasures in a.*

Then $B_{(1)}$ is obtained by having exactly three erasures from $A_{(1)}$, and $B_{(2)}$ is obtained by having exactly three erasures from $A_{(2)}$.

Proof. Since the erasures occur as three bursts of length 2 in a, each burst affects two consecutive positions in a.

Each position in a corresponds to either $A_{(1)}$ or $A_{(2)}$ based on its index. Specifically:

- Odd-indexed bits in a (affected by erasures) contribute to $A_{(1)}$,
- Even-indexed bits in a (affected by erasures) contribute to $A_{(2)}$.

Since each burst of length 2 in a affects exactly one odd-indexed bit and one even-indexed bit, the total of three bursts results in exactly three erasures in $A_{(1)}$ and three erasures in $A_{(2)}$. Therefore, the resulting matrix B satisfies the property that $B_{(1)}$ and $B_{(2)}$ have three erasures each, corresponding to their respective rows in A. □

With the above lemma, we conclude that if we have an efficient three erasure correcting code, we can apply this code twice to correct the 6 erasures from $X_{(2)}$ to $X_{(t)}$. As given in Lemma 2.5, Dumer provides a code with designed distance 4 with redundancy $1.5 \log_q n$, where $q = 2^{O(\log \log n)}$ is the alphabet size.

Let $S_3(x)$ be the function defined in Lemma 2.4 where $k = 3$. We can apply $S_3(x)$ to each block from $X_{(2)}$ to $X_{(t)}$ to get syndromes that store the necessary information in each block that helps to correct three deletions. Denote by $X_{i,j}$ the j-th block of row i and denote these syndromes as $S_3(X_{2,c_1})$, $S_3(X_{2,c_2})$, $\ldots$, $S_3(X_{t,c_{\frac{n}{Pt}}})$. We then concatenate all syndromes of the blocks from the same block column together and denote them as $\overline{S}_3(X_c)$, i.e. $\overline{S}_3(X_1) = (\overline{S}_3(X_{2,c_1})\overline{S}_3(X_{3,c_1}) \ldots \overline{S}_3(X_{t,c_1}))$.

We first consider the syndrome concatenations from odd-indexed columns and encode each concatenation as a symbol in Dumer's code of designed distance 4 with alphabet size $q_{D_4} = 2^{O(t \log \log n)}$. To generate these syndromes, we use the redundancy function associated with the code in Lemma 2.6. Define the **syndrome function**

$D_4(x)$ to be the function that maps a sequence to its redundancy symbols in the code $V_q^m(I_4, P^{m-1})$ from Lemma 2.6. These redundancy symbols serve as compressed information that allows us to recover erased values.

Encoding Process To recover erased values, we use a code with designed distance 4, which can correct up to 3 erasures, define:

$$g_1(x) = D_4(\overline{S}_3(X_1), \overline{S}_3(X_3), \ldots, \overline{S}_3(X_{\frac{n}{Pt}-1})),$$
$$g_2(x) = D_4(\overline{S}_3(X_2), \overline{S}_3(X_4), \ldots, \overline{S}_3(X_{\frac{n}{Pt}})).$$

By ensuring an appropriate choice of q and m and applying **shortening** where necessary, we integrate Dumer's code efficiently into our construction, providing a systematic and scalable approach to correcting three bursts of exactly t deletions.

Erasure Correction with Dumer's Code As in previous constructions, we must determine suitable values of q and m such that:

$$\frac{q^m - 1}{q - 1} - \frac{3m}{2} \geq \frac{n}{Pt}.$$

which leads to $m \simeq \log_q \frac{n}{Pt}$.

Since increasing m leads to an exponential increase in code length, we choose the smallest even m that meets this condition. If the resulting code length exceeds $\frac{n}{Pt}$, we apply shortening by fixing S positions to predetermined values before computing the syndrome, ensuring the final code length matches our target.

Now we can give the final construction of our code C_3 that corrects three bursts of

exactly t deletions.

3.3.3 Construction

To construct our code that corrects three bursts of t deletions, we will follow the following steps:

1. Matrix Representation: The q-ary sequence x of length n is represented as a $t \times \frac{n}{t}$ matrix X, where each row corresponds to a subsequence of x. Each row $X_{(i)}$ is further divided into blocks of length $P = \log n$.

2. Restriction in Run Length: To enable effective correction using Dumer's Code, we restrict the run length limit the first row $X_{(1)}$ to be $\mu(X_{(1)}) \leq \log n$, which means the length of the longest run in the first row $X_{(1)}$ is at most $\log n$.

3. Error Localization in the First Row: We apply syndrome compression, which can correct 3 deletions in a q-ary sequence, to the first row $X_{(1)}$. This identifies approximate positions of the three deletions.

4. Error Correction in Remaining Rows: Using the approximate deletion positions in $X_{(1)}$, we identify at most 6 affected blocks across the remaining rows. These blocks are treated as containing erasures. Instead of considering it as a 6 erasure problem, we treat it as a three bursts of 2 erasures. This allows us to use a code with distance 4 (Dumer's Code) twice, which efficiently handles up to 3 erasures. This eventually solves the 6 erasure problem.

The overall code is defined as follows:

Construction 3.9. Suppose $q \geq 2$, $n \geq 3t$ and $t \geq 1$. Let $g_1(x)$, and $g_2(x)$ be defined as above. Let $N_1 = 2^{12 \log n + o(\log n)}$ and $N_2 = 2^{1.5 \log n}$, for $0 \leq a \leq N_1$ and

$0 \leq b_1, b_2 \leq N_2$, define the code C_3 as:

$$C_3 \triangleq \left\{ X \in \Sigma_q^n : \begin{array}{r} \mu(X_{(1)}) \leq \log n \\ S_3(X_{(1)}) = a, \\ g_1(x) = b_1, g_2(x) = b_2 \end{array} \right\}$$

Then C_3 is a code correcting three bursts of exactly t deletions, computable in linear time, with redundancy at most $15 \log n + o(\log n)$.

The total redundancy of this code is computed as follows:

- Redundancy for Error Localization: Using syndrome compression (Sima, Gabrys, and Bruck 2020b) adds a redundancy of $12 \log n + o(\log n)$.
- Redundancy for Erasure Correction: According to Lemma 2.6, the total redundancy for correcting three erasures is at most $\frac{3}{2}m \sim 1.5 \log n$. Thus the total redundancy of correcting the remaining rows is $2 \times 1.5 \log n = 3 \log n$.

Summing these, the overall redundancy of the code is at most $15 \log n + o(\log n)$. This redundancy is higher than directly using syndrome compression (Sima, Gabrys, and Bruck 2020a), but our approach has the advantage of lower encoding and decoding time complexity when t is large.

To calculate the encoding and decoding time complexity of C_3, we need to calculate them step by step. Below we have two lemmas for the complexity and we prove their correctness.

Lemma 3.10. *The encoding time complexity of C_3 is $O(n^7)$.*

Proof. The encoding time complexity of syndrome compression is $O(n^{2b+1})$ when b is the number of deletions (Sima, Gabrys, and Bruck 2020a). Here we use syndrome compression in the first row $X_{(1)}$ and the number of deletions is 3, so our encoding time complexity of using syndrome compression in $X_{(1)}$ is $O(n^7)$. For the remaining rows, we first need the complexity of calculating syndromes in each block. Since every block is of length $O(\log n)$, the encoding complexity is $(\log n)^7$. Finally, we need to calculate the encoding complexity of Dumer's code. In Dumer 1995 it is stated that this is a linear code, and so the encoding complexity is $O(n)$. Combining these together, the overall encoding time complexity of C_3 is $O(n^7) + O(n) + (\log n)^7 = O(n^7)$. □

Lemma 3.11. *The decoding time complexity of C_3 is $O(n^7)$.*

Proof. The decoding time complexity of syndrome compression is $O(n^{2b+1})$ when b is the number of deletions (Sima, Gabrys, and Bruck 2020a). Here we use syndrome compression in the first row X_1 and the number of deletions is 3, and so our decoding time complexity of using syndrome compression in X_1 is $O(n^7)$. For the remaining rows, we first need the decoding complexity of each block. Since every block is of length $O(\log n)$, the decoding complexity is $(\log n)$. Finally, we need to calculate the decoding complexity of Dumer's code. In Dumer 1995 it is stated that the decoding complexity is $O(rn)$ where r is the redundancy of the code and n is the length of the code. In C_3, the redundancy of using Dumer's code with designed distance 4 twice is $2 \times 1.5 \log n$, and the code length is $\frac{n}{Pt}$. So the complexity is $O(n \log n)$. Combining these together, the overall time complexity of C_3 is $O(n^7) + (\log n) + O(n \log n) = O(n^7)$. □

Note that the time complexity of the encoder and decoder of C_3 is dominated by the time complexity of syndrome compression in the first row, which is $O(n^7)$. If

we directly use syndrome compression to construct codes correcting three bursts of deletions of length t, the time complexity is $\Omega(n^7 q^t)$ where q is the alphabet size. Therefore, C_3 has lower time complexity than using syndrome compression directly when t is large.

Chapter 4

Discussion

This chapter reflects on the proposed codes, analyzing their advantages and limitations compared to prior work. We discuss the practical applications of these codes in data storage, communication, and bioinformatics, and highlight potential future directions for extending and improving the constructions.

4.1 Extensibility of the Proposed Codes

The constructions presented in this thesis focus on correcting two or three bursts of exactly t deletions. However, the methodology extends naturally to larger values of k, allowing for the correction of any number of bursts. This extensibility stems from the modular structure of the proposed codes, where the first row is responsible for error localization and the remaining rows are handled as an erasure correction problem. In this section, we discuss how the framework scales when the number of bursts k or the burst length t increases.

4.1.1 Extending to Larger k: Correcting More Bursts of Exactly t Deletions

The primary challenge in extending the proposed construction to more than three bursts lies in efficiently locating the deletions in the first row of the matrix representation. For two or three bursts, we employed specialized deletion-correcting codes (such as Guruswami and Håstad's code or syndrome compression) to determine the approximate locations of the missing symbols. When k grows, the first row must be corrected using a k-deletion correcting code.

Syndrome Compression for Large k

A natural choice for handling a larger number of bursts is syndrome compression, which provides an efficient way to encode and recover sequences that have undergone multiple deletions. This ensures that the complexity of the first-row recovery remains manageable even for large k, preventing an exponential blow-up in computation.

Erasure Correction for the Remaining Rows

Once the approximate deletion locations are identified, the remaining rows of the matrix must be corrected using an erasure-correcting code. Although our previous constructions relied on Dumer's code with distance 4 or 5, extending to higher k requires a more flexible approach:

- **Reed-Solomon Code** can be directly applied to handle the erasures in the remaining rows. While Reed-Solomon codes may not be the most efficient choice due to their general nature, they provide a robust, well-established method for

handling a large number of erasures.

- **Repeated Application of Dumer's Code:** An alternative approach is to use Dumer's codes, which are available for designed distances $4, 5, 6$ etc. For example, if we need to correct 50 erasures (arising from 25 bursts of deletions), we can repeatedly apply a Dumer's code with designed distance 6, correcting 5 erasures at a time.

  However, repeatedly applying Dumer's code in this way starts to resemble treating each block independently rather than leveraging global structure across the matrix. This reduces efficiency and may lead to higher redundancy compared to more structured approaches. Thus, while Dumer's codes are useful for moderate values of k, a hybrid approach that strategically combines them with global erasure-correcting techniques may be more optimal for large k.

- **Specialized burst erasure codes:** Since we are not dealing with arbitrary $2k$ erasures but rather k bursts of exactly two erasures each, there is potential for designing more specialized codes that better exploit this structure.

Thus, the proposed approach naturally scales to an arbitrary number of bursts by modifying the first-row deletion correction mechanism and appropriately choosing an erasure-correcting code for the remaining rows.

4.1.2 Impact of Large t on Complexity and Redundancy

Unlike increasing k, which requires adjustments to the first-row correction and erasure correction strategy, increasing t has a minimal effect on the redundancy and computational complexity of the proposed codes.

- **Fixed Redundancy and Complexity:** Since our approach fundamentally transforms the problem of correcting bursts of deletions into a combination of deletion and erasure correction, the redundancy remains dependent on n and k, but not directly on t. The same erasure-correcting methods remain applicable regardless of t, making the approach scalable.

- **Computational Complexity of Syndrome Compression:** The main concern with large t arises in the complexity of syndrome compression, which is $O(n^7 q^t)$. This dependence on q^t becomes prohibitive when t grows large. However, the structured nature of the deletions suggests that alternative approaches, such as applying additional structure to the syndrome compression technique, may yield more computationally efficient solutions.

- **Advantage Over Direct Syndrome Compression for Large t:** Since our method applies syndrome compression only to the first row and not the entire sequence, the complexity does not increase as drastically as it would if syndrome compression were used directly for correcting all deletions in the sequence. This makes our approach more practical for large t compared to direct syndrome compression.

Summary of Extensibility

The modular design of our codes allows them to be extended naturally to handle larger numbers of bursts. By replacing the first-row deletion correction with a more general k-deletion correcting code (such as syndrome compression) and adapting the erasure correction step, our approach can be scaled to arbitrary k without increasing redundancy beyond $O(k \log n)$. Additionally, the method remains effective for

large t, with the primary limitation being the computational complexity of syndrome compression, which could be further optimized in future research.

4.2 Comparing with Ye et al.'s Work

The construction of the code C_2 for correcting two bursts of exactly t deletions in q-ary sequences is a key contribution of this thesis. While our approach demonstrates clear strengths in extensibility, it also comes with trade-offs when compared to prior work, particularly the code proposed by Ye, Yu, and Elishco (2024). Below, we discuss the advantages and limitations of C_2 in detail.

4.2.1 Advantages of C_2: Extensibility

- Extensive Design:

  The code C_2 integrates existing efficient error-correcting tools such as Liu et al.'s q-ary two-deletion correcting code (Liu, Tjuawinata, and Xing 2024) for error localization and Dumer's distance-5 code (Dumer 1995) for erasure correction. This modularity ensures that the code is easy to understand, analyze, and extend.

  In contrast, Ye et al.'s code relies on a specially designed erasure-correcting code tailored for their specific construction. While this design achieves lower redundancy, it is inherently less general and more difficult to adapt to scenarios involving more bursts of deletions.

- Extensibility to More Bursts:

  C_2 is extensively structured in a way that can be extended to cases involving

more than two bursts of deletions. This extensibility is due to the use of modular components like matrix representation, regularity conditions, and block-based error correction.

Ye et al.'s construction, while efficient for two bursts, does not generalize naturally to three or more bursts due to the limitations of their erasure-correcting strategy.

- Efficient Error Localization:
  The use of q-ary deletion-correcting codes in the first row of the matrix representation enables efficient identification of approximate deletion positions. This localization process is robust across different lengths of t and provides a foundation for correcting more bursts in future extensions.

4.2.2 Limitations of C_2: Redundancy

- The redundancy of C_2 is $7.4 \log n + 10 \log \log n + 3 \log q + O(1)$, which is higher than the redundancy of $5 \log n + O(\log \log n)$ achieved by Ye et al.'s code.

- In C_2, the use of syndromes for each block introduces additional redundancy, particularly when the number of blocks increases as t becomes smaller or n becomes larger. While this is mitigated by the logarithmic nature of the overhead, it still contributes to the overall redundancy gap compared to Ye et al.'s (Ye, Yu, and Elishco 2024) approach.

4.2.3 Balancing Redundancy and Practicality

Although Ye et al.'s code achieves lower redundancy, its lack of extensibility limits its application to scenarios involving more bursts or larger values of t. In contrast, C_2 offers a practical framework that is both modular and adaptable. This trade-off suggests that C_2 is better suited for applications where extensibility and generality are prioritized over achieving the absolute minimum redundancy.

4.3 Comparing with Syndrome Compression

The construction of C_3, a code that corrects three bursts of exactly t deletions, represents a significant step toward addressing more complex error patterns. In this section, we evaluate the advantages and disadvantages of C_3, particularly when compared to syndrome compression-based approaches (Sima, Gabrys, and Bruck 2020a), which, while not specifically tailored to burst deletions, are general enough to handle any set of deletions..

4.3.1 Advantages of C_3: Lower Complexity and Structured Design

- Lower Decoding Complexity for Large t:

  One of the most significant advantages of C_3 is its lower time complexity when t is large. This code has an encoding and decoding time complexity of $O(n^7)$. Syndrome compression, while achieving lower redundancy, has a time complexity of $O(n^7 q^t)$, which grows rapidly with t, making it practically infeasible for large values of t.

In contrast, C_3 leverages modular error correction steps that decouple the error localization and correction processes, resulting in significantly lower computational overhead during decoding.

- Efficient Handling of Bursts as Erasures:

  C_3 treats the affected blocks in the matrix as a sequence of erasures, reducing the problem to a combination of multiple erasure corrections. By encoding each block's syndrome and using Dumer's distance-4 code twice, C_3 efficiently manages the three bursts of deletions as six erasures without requiring specialized erasure-correcting designs.

- Extensibility and Modularity:

  Similar to C_2, the design of C_3 is extensive and modular. This makes it easy to analyze, implement, and potentially extend to handle even more bursts of deletions in the future.

  The code also uses a general-purpose matrix representation and block-based error correction strategy, avoiding the reliance on highly specific error-correcting mechanisms.

4.3.2 Limitations of C_3: Higher Redundancy

- Higher Redundancy:

  The redundancy of C_3 is at most $15 \log n + o(\log n)$, , which is higher than the redundancy of approximately $12 \log n + o(\log n)$ achieved by direct syndrome compression. This difference primarily arises from the use of modular components and the additional overhead introduced by treating the problem as a series of erasure corrections.

- Additional Syndromes for Block-Level Correction:

  C_3 relies on the computation and storage of block-level syndromes for all affected rows in the matrix representation. While this approach simplifies the correction process, it adds a layer of redundancy that does not exist in direct syndrome compression methods.

4.3.3 Balancing Complexity and Redundancy

The key trade-off between C_3 and syndrome compression lies in balancing computational complexity and redundancy. Syndrome compression is the best we know for three bursts of t deletions but suffers from high decoding complexity, particularly for large t. C_3, while incurring higher redundancy, provides a practical alternative with significantly reduced computational requirements, making it better suited for scenarios where decoding efficiency is critical.

4.4 Practical applicability and use cases

This thesis focuses on the theoretical construction and analysis of burst deletion correcting codes, but the potential applications of these codes in real-world scenarios highlight their practical importance. Burst deletion correcting codes are especially relevant in domains where clustered errors are common, such as data storage systems, communication networks, and bioinformatics. Below, we outline several key applications.

4.4.1 Data Storage Systems

Modern storage devices, such as hard drives and optical media, often encounter burst errors due to physical defects, wear, or operational failures (Bang et al. 2023). Burst deletions can occur when a sequence of consecutive bits or symbols is erased during read/write processes, making burst deletion correcting codes like C_2 and C_3 valuable for ensuring reliable data recovery. These codes are particularly useful in high-reliability applications, such as enterprise data centers or archival storage, where both redundancy and efficient decoding are critical.

For solid-state drives (SSDs), errors are more commonly in the form of erasures, where a cell becomes unreadable but its location remains identifiable (Kishani, Ahmadian, and Asadi 2020). While burst deletion correcting codes may not directly address such scenarios, the adaptability of these methods could inspire hybrid techniques to handle mixed error models that combine deletions and erasures.

Additionally, DNA data storage—a promising technology for high-density and long-term data storage—presents unique challenges, including burst errors caused by synthesis or sequencing processes. The adaptability of the proposed codes to q-ary alphabets makes them suitable for improving error correction in DNA data storage systems, thereby enhancing data integrity.

4.4.2 Communication Systems

Burst deletion errors frequently occur in wireless communication due to interference, packet loss, or synchronization issues (Yi et al. 2023). They are also common in wired systems experiencing localized noise or physical damage.

The modular and computationally efficient design of the proposed codes makes them suitable for real-time error correction in systems like satellite communications, mobile networks, and underwater communication. Since the codes are adaptable to q-ary sequences, they are also applicable to advanced communication schemes such as high-order modulation formats used in 5G and beyond.

4.4.3 Relevance to Broader Applications

Although this thesis is grounded in theoretical computer science, the potential use cases of the proposed codes demonstrate their broader relevance. These codes offer a flexible and efficient approach to addressing burst deletion errors in a variety of domains. By bridging theoretical constructs with practical needs, this work lays the foundation for further exploration and development in both research and applied settings.

4.5 Future Directions

While this thesis presents progress in the design of burst deletion correcting codes, there remain several opportunities for future research to extend and refine the ideas introduced here. Two promising directions are discussed below.

4.5.1 Codes for Multiple Bursts of 2 Erasures

In the current work, correcting bursts of length t deletions in a sequence often involves handling $2t$ erasures in the intermediate stages of decoding. However, these erasures are not independent but are instead clustered into t bursts of length 2. By leveraging

the structured nature of these erasure patterns, future research could aim to construct specialized codes that handle such scenarios more efficiently.

Key aspects for exploration include:

- Exploiting Structure:
  Developing codes that specifically target clustered patterns of erasures rather than treating them as independent errors. This would likely lead to reduced redundancy.

- Extending Existing Techniques:
  Adapting or extending methods such as Dumer's code or syndrome compression to directly address the bursty nature of erasures.

- Applications:
  These new codes could have significant utility in communications, data storage, and other domains where burst errors are prevalent.

By constructing efficient codes tailored to this specific case, it may be possible to achieve significant reductions in both redundancy and decoding complexity.

4.5.2 Codes for Three or More Bursts of Exactly t Deletions

This thesis provides a code that can correct three bursts of t deletions, with a redundancy of at most $15 \log n + o(\log n)$, and demonstrates its efficiency in terms of decoding complexity. However, there is room for improvement in both redundancy and complexity, particularly for sequences with more than three bursts.

Future research could focus on:

- Lower Redundancy:

  Investigating new approaches that reduce redundancy while maintaining efficient decoding.

- Scalability to More Bursts:

  Extending the current framework to correct four or more bursts of exactly t deletions with competitive redundancy and time complexity.

These efforts could further push the boundaries of burst deletion correcting codes and provide a foundation for addressing increasingly complex error patterns in practical systems.

Chapter 5

Conclusion

This thesis addresses the challenge of correcting burst deletions in binary and q-ary sequences, an important problem in error-correcting codes with applications in data storage and communication. Specifically, we focused on constructing codes that correct two or three bursts of exactly t deletions while balancing redundancy and decoding complexity.

We presented four key contributions:

- Code C_1: A Code for Two Bursts of t Deletions in Binary Sequences.
  This code achieves a redundancy of $6.4 \log n + 10 \log \log n + O(1)$ and offers an efficient approach for correcting burst errors in binary sequences. It serves as a foundation for the more general constructions presented in this thesis.

- Code C_2: A Code for Two Bursts of t Deletions in q-ary Sequences.
  This code achieves redundancy of $7.4 \log n + 10 \log \log n + 3 \log q + O(1)$ and demonstrates extensibility and modularity, making it extensible to scenarios involving more bursts of deletions.

- Code C_3: A Code for Three Bursts of t Deletions.
  This construction achieves a redundancy of $15 \log n + o(\log n)$ and provides a significant reduction in decoding complexity compared to syndrome compression codes, especially for large t.

- Theoretical Insights and Practical Implications.

  We analyzed the redundancy and complexity trade-offs of our constructions, showing that while C_2 and C_3 have higher redundancy than some prior work, their extensive design and lower complexity make them valuable for practical applications.

The implications of this research extend beyond theoretical advancements. Burst deletion correcting codes have potential use cases in domains such as data storage, communication systems, and genome sequencing for DNA data storage, where clustered errors frequently occur. Our work provides a foundation for exploring efficient, scalable, and adaptable solutions to handle such errors.

Despite these achievements, certain limitations remain. For instance, C_2 has a higher redundancy than Ye et al.'s code, and C_3 could benefit from further optimization to reduce its redundancy or extend its applicability to more bursts of deletions. Addressing these challenges represents exciting directions for future research.

In conclusion, this thesis contributes significantly to the study of burst deletion correcting codes, offering new constructions that balance redundancy and complexity. By addressing the theoretical and practical aspects of burst error correction, this work provides a valuable framework for further exploration in error-correcting codes.

Chapter 6

Summary

This thesis explores the design and analysis of burst deletion correcting codes, addressing a challenging error scenario in coding theory. Through extensive and modular constructions, this work provides new approaches for correcting bursts of deletions in both binary and q-ary sequences.

We proposed three codes: one for two bursts of exactly t deletions in binary, one for q-ary sequences, and one for three bursts of exactly t deletions. These codes balance redundancy, complexity, and extensibility, providing practical frameworks for real-world applications such as data storage and communication. While our constructions sometimes incur higher redundancy than existing methods, they excel in extensibility, computational efficiency, and adaptability to larger error scenarios.

By advancing the understanding of burst deletion correction, this thesis lays a foundation for future research in constructing codes with lower redundancy, improving complexity, and addressing even more complex error models. The findings presented here contribute both to the theoretical development of error-correcting codes and to their application in domains requiring robust data reliability.

Bibliography

Varshamov, R. R. and G. M. Tenengolts (1965). "Code Correcting Single Asymmetric Errors (in Russian)". In: *Avtomat. i Telemekh.* 26.2, pp. 288–292.

Dumer, I. (1995). "Nonbinary double-error-correcting codes designed by means of algebraic varieties". In: *IEEE Transactions on Information Theory* 41.6, pp. 1657–1666. DOI: 10.1109/18.476238.

Gregori, S. et al. (2003). "On-chip error correcting techniques for new-generation flash memories". In: *Proceedings of the IEEE* 91.4, pp. 602–616. DOI: 10.1109/JPROC.2003.811709.

Mercier, Hugues, Vijay Bhargava, and Vahid Tarokh (Jan. 2010). "A Survey of Error-Correcting Codes for Channels With Symbol Synchronization Errors". In: *Communications Surveys Tutorials, IEEE* 12, pp. 87–96. DOI: 10.1109/SURV.2010.020110.00079.

Venkataramanan, Ramji, Hao Zhang, and Kannan Ramchandran (2010). "Interactive low-complexity codes for synchronization from deletions and insertions". In: *2010 48th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 1412–1419. DOI: 10.1109/ALLERTON.2010.5707079.

Schoeny, C., F. Sala, and L. Dolecek (2017). "Novel combinatorial coding results for DNA sequencing and data storage". In: *Proceedings of the IEEE Asilomar Conference on Signals, Systems, and Computers*. Asilomar, CA, USA, pp. 511–515. DOI: 10.1109/ACSSC.2017.8335742.

Schoeny, C., A. Wachter-Zeh, et al. (Apr. 2017). "Codes correcting a burst of deletions or insertions". In: *IEEE Transactions on Information Theory* 63.4, pp. 1971–1980. DOI: 10.1109/TIT.2017.2661747.

Schoeny, Sala, and Dolecek (2017). "Novel combinatorial coding results for DNA sequencing and data storage". In: *2017 51st Asilomar Conference on Signals, Systems, and Computers*, pp. 511–515. DOI: 10.1109/ACSSC.2017.8335392.

Field, M. A. et al. (2019). "Recurrent miscalling of missense variation from short-read genome sequence data". In: *BMC Genomics* 20.Suppl 8, p. 546. DOI: 10.1186/s12864-019-5863-2.

Sima, J., N. Raviv, and J. Bruck (2019). "Two deletion correcting codes from indicator vectors". In: *IEEE Transactions on Information Theory* 66.4, pp. 2375–2391.

Kishani, Mostafa, Saba Ahmadian, and Hossein Asadi (2020). "A Modeling Framework for Reliability of Erasure Codes in SSD Arrays". In: *IEEE Transactions on Computers* 69.5, pp. 649–665. DOI: 10.1109/TC.2019.2962691.

Sima, J., R. Gabrys, and J. Bruck (2020a). "Optimal systematic t-deletion correcting codes". In: *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, pp. 769–774. DOI: 10.1109/ISIT44484.2020.9173986.

— (2020b). "Syndrome compression for optimal redundancy codes". In: *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, pp. 751–756. DOI: 10.1109/ISIT44484.2020.9174009.

Guruswami, Venkatesan and Johan Håstad (Aug. 2021). "Explicit two-deletion codes with redundancy matching the existential bound". In: *IEEE Transactions on Information Theory* 67.8, pp. 5605–5611. DOI: 10.1109/TIT.2021.3078981.

Bang, Jiwoo et al. (2023). "Design and Implementation of Burst Buffer Over-Subscription Scheme for HPC Storage Systems". In: *IEEE Access* 11, pp. 3386–3401. DOI: 10.1109/ACCESS.2022.3233829.

Yi, Chen et al. (2023). "Correcting Non-Binary Burst Deletions/Insertions with De Bruijn Symbol-Maximum Distance Separable Codes". In: *IEEE Communications Letters* 27.8, pp. 1939–1943. DOI: 10.1109/LCOMM.2023.3286448.

Liu, Shu, Ivan Tjuawinata, and Chaoping Xing (June 2024). "Explicit Construction of q-Ary 2-Deletion Correcting Codes With Low Redundancy". In: *IEEE Transactions on Information Theory* 70.6, pp. 4093–4101. DOI: 10.1109/TIT.2024.3360964.

Ye, Z., W. Yu, and O. Elishco (2024). *Codes correcting two bursts of exactly b deletions.* arXiv preprint arXiv:2408.03113.