Correcting Errors in DNA Storage

Thesis by
Jin Sima

In Partial Fulfillment of the Requirements for the
Degree of
Doctor of Philosophy

CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2022
Defended May 25, 2022

ACKNOWLEDGEMENTS

My Phd journey is a colorful, joyful experience and valuable treasure in my life. For me, the most significant part of this experience is the interaction with many great people that I am thankful to, though words understate my gratitude and what they helped me. It is hard to include all pieces of the nice memories I have during my Phd life, simply because there are so many.

I am deeply grateful to my advisor, Professor Jehoshua (Shuki) Bruck, for his endless support and guidance, in many aspects including research, career, and life. What an incredibly amazing advisor means to a Phd is self-evident and Shuki means more. Shuki's optimism, wisdom, caring for students, pursuit of perfection, and inspiring ways to conduct original and fundamental research are what I always benefited and will benefit from and what have shaped me during the past years. What he had taught me is far beyond producing this thesis.

I am grateful to my thesis and candidacy committee, Professor Babak Hassibi, Professor Victoria Kostina, Professor Leonard Schulman, Professor Anxiao (Andrew) Jiang, and Professor Netanel Raviv, who are always willing to help and provided insightful feedback to this thesis. I learned from the classes of Professor Babak Hassibi, Professor Victoria Kostina, and Professor Leonard Schulman, and wish that I could take those classes again and take more.

I would also like to express my gratitude to my collaborators, Professor Netanel Raviv, who appears in many places in this acknowledgement because of the many roles he played and his help in my Phd study, Dr. Ryan Gabrys and Professor Fazad Farnoud, who are friend and alumni of the Paradise lab, Professor Shao-Lun Huang, who kindly offered me an opportunity to visit his group, Feng Zhao, Shuche Wang, and Kordag Kilic, who is my labmate, for kindly sharing their ideas and being nice to work with.

I am also grateful to alumnus and frieds of the Paradise lab, Professor Anxiao (Andrew) Jiang, Professor Moshe Schwartz, Professor Mathew Cook, Professor Hongchao Zhou, and Professor Anatoly Khina, for sharing many inspiring and nice ideas and the insightful discussions, though I did not have a chance to closely work with them. The Paradise lab is like a family and I am grateful to past and present labmates, Wentao Huang, Yue Li, Netanel Raviv, Siddharth Jain, Bijan Mazaheri, and Kordag Killic, who are all nice and with whom I had a wonderful time and fond memories of staying in the lab.

Friendship has been a source of happiness and support during my Phd life. I am grateful to my friends Hao Zhou, Songtai Li, Wen-loong Ma, Honglie Ning, Fangzhou Xiao, Xiangxiang Xu, Jian Xu, Fengyu Zhou, Chen Liang, Peida Tian, Xiang Li, Yiqiu Ma, Junyi Shan, Zhe Jia, Zhichao Shen, Liyin He, Jiaqing Wang, Yida Li, Yangcheng Luo, Zhi Ren, Changnan Peng, Xiaotian (Jim) Zhang, for their support, help, and the pleasant time I had with them during my phd study.

Finally, I owe my deepest gratitude to my parents, Chang Sima and Meilan Ye, who gave me the opportunity to experience all these delightful moments and have been supporting me with their immeasurable love, tolerance, and encouragement all along.

ABSTRACT

DNA-based storage has potentially unprecedented advantages of high information density and long duration, and is one of the promising techniques to meet the ever-growing demands to keep data in the future. As noise and errors are present in almost every procedure during reading, writing, and storing of information in DNA storage systems, error correction is inevitable to guarantee reliable data storage in DNA. Moreover, it is often required that error correction is done in an efficient manner to reduce the cost and time needed for reading and writing data. Due to the technology constraints and physical limitations, error correction in DNA-based storage poses the following challenges that differ from those in traditional digital data transmission and storage systems.

1. A combination of deletion, insertion, and substitution errors present. The goal is to construct efficient codes correcting these errors. While substitution errors are special cases of deletion and insertion errors, and are well studied under the current theory and practice frameworks, deletion and insertion errors are much more difficult to deal with, and less understanding was gained for deletion and insertion errors.

2. Error correction is over an unordered set of strings, rather than over a single string, which can be regarded as a set of ordered strings. The latter, which includes the above deletion/insertion coding problem, is commonly studied for current digital communication and storage systems. Our goal is to extend the deletion/insertion correction capability for a single string to a set of unordered strings.

3. The decoder observes multiple noisy copies of every coded string. The problem is to deduce a set of strings (or a single string) from a collection of their noisy samples, also studied as the population recovery (or trace reconstruction for a single string) problem. The problem is well answered with substitution errors only and becomes elusive with the introduction of deletion and insertion errors.

This thesis tries to address the above challenges. For the first challenge, we proposed binary codes correct any constant number of deletions and/or insertions with order-wise optimal redundancy, which made a step toward a solution to a longstanding open problem introduced by Levenshtein in 1960s. We also extended it to different settings, in particular, non-binary deletin/insertion correcting codes suitable for DNA storage applications.

For the second challenge, we established lower and upper bounds on the optimal redundancy of codes correcting any number of substitution, deletion, and insertion errors and found that the redundancy needed for coding over an unordered set of strings is order-wise the same as that needed for coding over a ordered set of strings. Using our results for the first challenge, we proposed codes correcting any constant number of deletion/insertion errors with order-wise optimal redundancy under some parametter settings.

For the third challenge, we studied the problem of trace reconstruction, which asks the number of noisy samples needed to reconstruct a single string. While there is a exponential gap between upper and lower bounds on sample complexities in general, we showed that a polynomial number of samples suffice, given a reference string that is within constant edit distance from the target string.

Apart from dealing with the above challenges, we investigated error correction for multi-head racetrack memory applications. The problem can be considered as correcting any constant number of deletions/insertions in a single string with multiple noisy copies, with the help of coding. Different from the settings we considered above in the trace reconstruction problem, where noisy copies are independent given the target string, in racetrack memory, the noisy copies are correlated, and the number of errors is small compared to the trace reconstruction problem. We derived a lower bound on redundancy and proposed a code correcting any number of deletions/insertions with order-wise optimal redundancy.

PUBLISHED CONTENT AND CONTRIBUTIONS

[1] J. Sima and J. Bruck. “Correcting k Deletions in Racetrack Memory”. In: *To be submitted* (2022).
Jin Sima participated in the conception of the project, derived the results, and wrote the manuscript.

[2] J. Sima, R. Gabrys, and J. Bruck. “Syndrome Compression and Codes Correcting Deletions in Binary and Non-binary Cases”. In: *To be submitted* (2022).
Jin Sima participated in the conception of the project, derived the main results, and participated in the writing of the manuscript.

[3] J. Sima, N. Raviv, and J. Bruck. “Robust Indexing – Optimal Codes for DNA Storage”. In: *To be submitted* (2022).
Jin Sima participated in the conception of the project, derived the results, and wrote the manuscript.

[4] J. Sima and J. Bruck. “Trace Reconstruction with Bounded Edit Distance”. In: *Proc. IEEE Int. Symp. Inf. Theory*. Melbourne, Australia: IEEE, 2021. DOI: 10.1109/ISIT45174.2021.9518244.
Jin Sima participated in the conception of the project, derived the results, and wrote the manuscript.

[5] J. Sima, N. Raviv, and J. Bruck. “On Coding Over Sliced Information”. In: *IEEE Trans. on Inf. Th* 67.5 (2021), pp. 2793–2807. DOI: 10.1109/TIT.2021.3063709.
Jin Sima participated in the conception of the project, derived the main results, and participated in the writing of the manuscript.

[6] J. Sima and J. Bruck. “On Optimal k-Deletion Correcting Codes”. In: *IEEE Trans. on Inf. Th* 67.6 (2020), pp. 3360–3375. DOI: 10.1109/TIT.2020.3028702.
Jin Sima participated in the conception of the project, derived the results, and wrote the manuscript.

[7] J. Sima, R. Gabrys, and J. Bruck. “Optimal Codes for the q-ary Deletion Channel”. In: *Proc. IEEE Int. Symp. Inf. Theory*. Los Angeles, CA, USA: IEEE, 2020. DOI: 10.1109/ISIT44484.2020.9174241.
Jin Sima participated in the conception of the project, partially derived the results, and participated in the writing of the manuscript.

[8] J. Sima, R. Gabrys, and J. Bruck. “Optimal Systematic t-Deletion Correcting Codes”. In: *Proc. IEEE Int. Symp. Inf. Theory*. Los Angeles, CA, USA: IEEE, 2020. DOI: 10.1109/ISIT44484.2020.9173986.
Jin Sima participated in the conception of the project, derived the results, and wrote the manuscript.

[9] J. Sima, R. Gabrys, and J. Bruck. “Syndrome Compression for Optimal Redundancy Codes”. In: *Proc. IEEE Int. Symp. Inf. Theory*. Los Angeles, CA, USA: IEEE, 2020. DOI: 10.1109/ISIT44484.2020.9174009.
Jin Sima participated in the conception of the project, partially derived the results, and participated in the writing of the manuscript.

[10] J. Sima, N. Raviv, and J. Bruck. “Robust Indexing - Optimal Codes for DNA Storage”. In: *Proc. IEEE Int. Symp. Inf. Theory*. Los Angeles, CA, USA: IEEE, 2020. DOI: 10.1109/ISIT44484.2020.9174447.
Jin Sima participated in the conception of the project, derived the results, and wrote the manuscript.

[11] J. Sima, N. Raviv, and J. Bruck. “Two Deletion Correcting Codes from Indicator Vectors”. In: *IEEE Trans. on Inf. Th* 66.4 (2020), pp. 2375–2391. DOI: 10.1109/TIT.2019.2950290.
Jin Sima participated in the conception of the project, derived the main results, and participated in the writing of the manuscript.

[12] J. Sima and J. Bruck. “Correcting Deletions in Multiple-Heads Racetrack Memories”. In: *Proc. IEEE Int. Symp. Inf. Theory*. Paris, France: IEEE, 2019. DOI: 10.1109/ISIT.2019.8849783.
Jin Sima participated in the conception of the project, derived the results, and wrote the manuscript.

[13] J. Sima and J. Bruck. “Optimal k-Deletion Correcting Codes”. In: *Proc. IEEE Int. Symp. Inf. Theory*. Paris, France: IEEE, 2019. DOI: 10.1109/ISIT.2019.8849750.
Jin Sima participated in the conception of the project, derived the results, and wrote the manuscript.

[14] J. Sima, N. Raviv, and J. Bruck. “On Coding over Sliced Information”. In: *Proc. IEEE Int. Symp. Inf. Theory*. Paris, France: IEEE, 2019. DOI: 10.1109/ISIT.2019.8849596.
Jin Sima participated in the conception of the project, derived the main results, and participated in the writing of the manuscript.

[15] J. Sima, N. Raviv, and J. Bruck. “Two Deletion Correcting Codes from Indicator Vectors”. In: *Proc. IEEE Int. Symp. Inf. Theory*. Vail, CO, USA: IEEE, 2018. DOI: 10.1109/ISIT.2018.8437868.
Jin Sima participated in the conception of the project, derived the main results, and participated in the writing of the manuscript.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

Chapter 1

INTRODUCTION

We are in an era of data explosion. It was estimated that the total amount of data generated and replicated in 2020 was 64.2 Zettabytes (1 Zettabyte = 10^{21} Byte), and this amount was projected to be 175 Zettabytes by 2025. Storing data size of this magnitude requires enormous space and high maintenance cost under current memory technologies such as flash, hard disk drives. In fact, only 6.8 Zettabytes of storage were deployed worldwide in 2020, according to [1]. As a result, most of the data were not saved.

One of the reasons for high maintenance cost of storage is that the lifespan of devices is at most ten years for flash memories and hard drives and up to thirty years for tape memories. Hence, the storage devices have to be replaced every three to twenty years, depending on the storage media and operation. In order to scale up the storage capacity at a rate comparable to the data growing rate, it is desirable to find a storage technology that is convenient and cheap to keep information.

Inspired by nature, people have been looking for storage solutions from DNA-based techniques, realizing that our chromosomes are actually a robust and energy efficient storage that records our biological information. In chromosomes, our biological information is encoded into sequences of nucleotides, each composed of one of the four bases A, C, G, and T, in a similar manner as we encode data into bits. DNA-based storage potentially has an unprecedented advantage in information density and duration. Ideally, the world's data could be stored in a coffee mug. As Richard P. Feynman put it, "There's plenty of the room at the bottom" [31]. In addition, information can be kept in DNA molecules for an incredibly long time, which is at least a hundred years, with low energy consumption.

There have been many successful attempts storing information in DNA molecules. The early ones date back to 1988, when thirty-five bits represented by a 5×7 binary matrix were encoded into a DNA molecule and inserted into *E. coli*, by Havard researchers [30]. Yet DNA-based storage of "considerable" size was not achieved until 2012–2013, where [24] and [36] stored 643KB and 739KB of data, respectively. These experiments have ignited the imagination of practitioners and theoreticians alike, and many works followed suit with various implementations [29, 99]. Up till now, the best size achieved was 200MB [73].

While DNA-based storage looks promising, several problems need to be addressed before its practical use. These problems include: (1) High cost and long time for synthesis and sequencing, which corresponds to data reading and writing, respectively, in DNA-based storage; (2) Random access that retrieves any specific part from a large pool of data; (3) Errors that are present in reading, writing, and storing processes in DNA-based storage. The first two affect the scalability of DNA storage. As technologies develop, it is reasonable to believe that the cost and time for synthesizing and sequencing will decrease by orders of magnitudes and more techniques will come out achieving efficient random access.

This thesis deals with the third problem, correcting errors, which plays a crucial role in delivering reliable DNA-storage systems. In addition, efficient error correcting methods help reduce the length and number of DNA molecules needed to store information, and thus reduce the time and cost for synthesising and sequencing. In the following, we present a model describing the current DNA-based storage and list some of the main challenges for error correction. Then, we give an overview of the contributions of this thesis.

1.1 DNA-Based Storage: Models

In DNA-based storage, data are encoded by a sequence of four bases of nucleotides, A, C, G, and T, and stored in synthesized DNA molecules. Fig. 1.1 shows the workflow of the whole reading/writing process, the details of which can be found in [73]. In the writing process, after encoding the data as a set of strings over a four-symbol alphabet (A, C, G, and T), the corresponding DNA molecules are synthesized (possibly with errors) and dissolved inside a solution. Due to the synthesizing and sequencing technology constraints, the length of each string, i.e., the number of bases in a string, is not large. The typical value of the string length is within the order of magnitude of 10^2.

When trying to retrieve the data, a chemical process called *Polymerase Chain Reaction* (PCR) is applied, which serves as the random access and drastically amplifies the number of copies of the DNA molecules that contain the desired data. In the reading process, the pool of DNA molecules is sampled and sequenced. The sequencing process reads the 4-ary DNA string contained in each sampled DNA molecule. Since the desired DNA molecules are amplified in the PCR process, it is highly possible that each ampliefied DNA molecule is sampled many times and

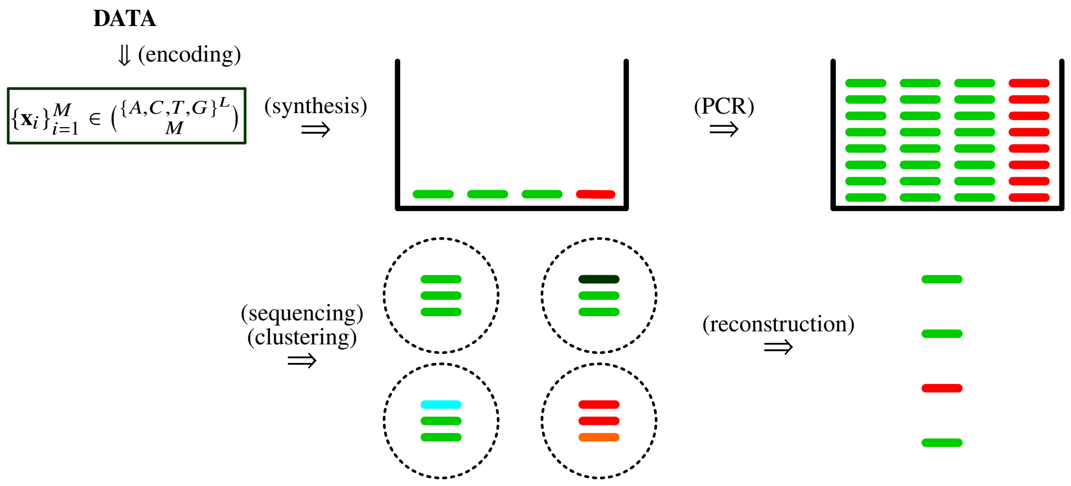

Figure 1.1: An illustration of a typical operation of a DNA storage system.

as a result, we obtain multiple (possibly erroneous) reads of the amplified strings. These reads are clustered according to their respective similarity. Then, sequence reconstruction algorithms are used within each cluster, in order to come up with the most likely origin of the reads in that cluster), which we consider as the string contained in an amplified DNA molecule. Finally, all reconstructed origins are collected and decoded back to data. The following gives an example of how strings are processed when going through the system described in Fig. 1.1.

Example 1.1.1. *Let $L = 3$ and $M = 3$. The data is first mapped into a set of strings, say, $\{ACT, AGG, TCG\}$. The synthesizing process produces three DNA molecules containing ACT, AGG, and TCC, respectively, where the string TCC is an erroneous version of TCG because of synthesis errors. The PCR process generates multiple copies of ACT, AGG, and TCC. After sampling those copies and sequencing them, one might get a set of strings $\{ACT, ACT, ACT, AGG, AGG, AGC, TCC, TCG, TCC, TCC\}$ and cluster them as $\{ACT, ACT, ACT\}$, $\{AGG, AGG, AGC\}$, and $\{TCC, TCG, TCC, TCC\}$. Then, the cluster origins are reconstructed as $\{ACT, AGG, TCC\}$, and decoded to data.*

1.2 Error Correction in DNA Storage: Challenges

If no noises or errors are introduced in the above procedures, it would not be hard to encode and decode the data, and the information can be stored reliably. However, errors are present by nature. In fact, errors occur in each of the above physical procedures. In synthesizing and sequencing processes, errors might come from limitations in technologies. In PCR, errors may come from unexpected chemical reactions. Even when storing those DNA molecules in the pool, mutations can happen. To successfully write and read the data, error correction schemes are needed.

Error correction techniques are common in current digital communication and storage systems. Among them, one of the major approaches is to construct error correcting codes, which map the data represented by sequences of bits to codewords, usually sequences with fixed alphabet size. It is required that a codeword can be recovered when some parts of it are affected by errors. Error correction codes have been deeply studied since the 1940s and many classical results such as Hamming code, Golay code, Bose–Chaudhuri–Hocquenghem (BCH) code, Reed-Muller (RM) code, Reed-Solomon (RS) code, Low-density parity check (LDPC) code, turbo code, and polar code, contribute to reliable digital communication and storage systems in the past and today [39].

In DNA-based storage, the special physical and technological constraints pose some new challenges to error correction, which are different from those in traditional digital systems. These challenges motivate us to study problem settings that are less studied in traditional information and coding theoretical works, and we believe these challenges are fundamental and might arise in current or future applications.

Deletion/Insertion Errors

In DNA-based storage, the basic types of errors are deletion, insertion, and substitution. When deletion errors occur, some bases are deleted, the locations of which are unknown. As a result, the DNA string becomes its subsequence. Similarly, under insertion errors, extra bases are inserted into the DNA string at unknown locations and the DNA string becomes its supersequence. Substitution errors replace some of the bases, the locations of which are unknown, in the string with other bases.

Example 1.2.1. *Let $ACCTGAT$ be a string of bases. If the third base C is deleted, the resulting string becomes $ACTGAT$. If a base A is inserted after the second base, the string becomes $ACACTGAT$. If the first base is replaced with T, the string becomes $TCCTGAT$. A combination of the three errors renders $TCATGAT$.*

The substitution errors are well studied, since the errors are "independent" and "memoryless," in the sense that substitutions at specific locations do not affect the bases at other locations. All of the above mentioned error correcting codes (Hamming code, Golay code, BCH code, etc.) were proposed for and are highly efficient in correcting substitution errors. On the other hand, deletion or insertion errors, also referred to as synchronization errors in some data transmission contexts, are less understood. One of the difficulties of addressing deletion and insertion errors is that they are "dependent" and not "memoryless." A deletion or insertion occurring at the beginning of a string affects the following part of the string. Moreover, a substitution error can be considered as a deletion error followed by a insertion error. Hence, substitution errors can be regarded as a special case of deletion and insertion errors.

Repeating the bases is one way of protecting strings from deletion and insertion errors, and is what nature does to protect our biological information in DNA molecules. However, this method is not efficient for information storage as it introduces many redundant bases. The search for efficient deletion/insertion correcting codes has been going on since 1960s. Yet, for more than fifty years, the only known efficient deletion/inserticorrecting code is capable of correcting a single deletion, though, there is recently much exciting progress going on towards a general solution.

Coding over Unordered Sets

As mentioned in Sec. 1.1, the data in a DNA storage system is stored as a pool of short strings that are dissolved inside a solution, and consequently, these strings are obtained at the decoder in an *unordered* fashion. Furthermore, the sampling method does not allow the decoder to count the exact number of appearances of each string in the solution, but merely to estimate relative concentrations. These restrictions have ignited the interest in *coding over sets* [62, 61, 83, 85, 89], a model that also finds applications in transmission over asynchronous networks, where different packets are routed along different paths of varying lengths, and are obtained in an unordered and possibly erroneous form at the decoder.

Different from traditional communication and storage channel models, where error correction coding encodes data into a single string, coding over sets is a channel model where data is encoded into a set of strings, usually of equal length and with no repeats. Most of current schemes, such as TCP/IP, use indexing prefix for each string (packet). While indexing schemes achieve Channel capacity [85] under some settings, these schemes require high indexing cost, when the number of strings is large and the string length is small (say, several hundreds), as in DNA storage applications. To the best of our knowledge,

Recover Strings from Noisy Copies

The aforementioned challenges (deletion/insertion errors and coding over unordered sets) pose problems to the encoder design, which is about writing data. When reading data, the sampling and sequencing procedures described in Sec. 1.1 produce multiple copies of each string. Because of the presence of errors, these copies are noisy. The goal of the decoder is to recover the set of strings, from a collection of noisy copies of the strings, and then decode the strings back to data. Note that for each noisy copy, it is not known which string generates this copy. This problem was also studied, with the name *population recovery*, which tries to identify the support of a distribution (the strings that are sampled with positive probability), from multiple noisy samples. The population recovery problem asks the number of samples needed to recover the support. With substitution errors as noises only, the population recovery problem was well studied, and a polynomial (with respect to the string length) number of samples are needed to recover the strings. However, when deletion and insertion errors present as noises, the problem becomes very difficult. In fact, it is even difficult when there is only a single support in the distribution, a problem called *trace reconstruction*.

Therefore, the current solutions take a step back, and decompose the population recovery problem into two independent tasks. The first is clustering, based on sample similarities, that aims to group the noisy samples such that the noisy samples within the same group come from the same string. The second is trace reconstruction, that aims to recover the string that generates the noisy samples within each group. The number of noisy samples needed to solve the trace reconstruction task is still an open problem with the presence of deletion/insertion errors. The state-of-the-art algorithms require exponential (with respect to the string length) number of samples, while we only know that at least a polynomial number of samples are needed.

We remark that a perfect population recovery algorithm at the decoder does not correct all errors and solve the problems at the encoder. The population recovery at the decoder can at best recover the strings after the PCR process. Since errors occur during the synthesizing, PCR processes, and even during storing information, error correcting codes are needed.

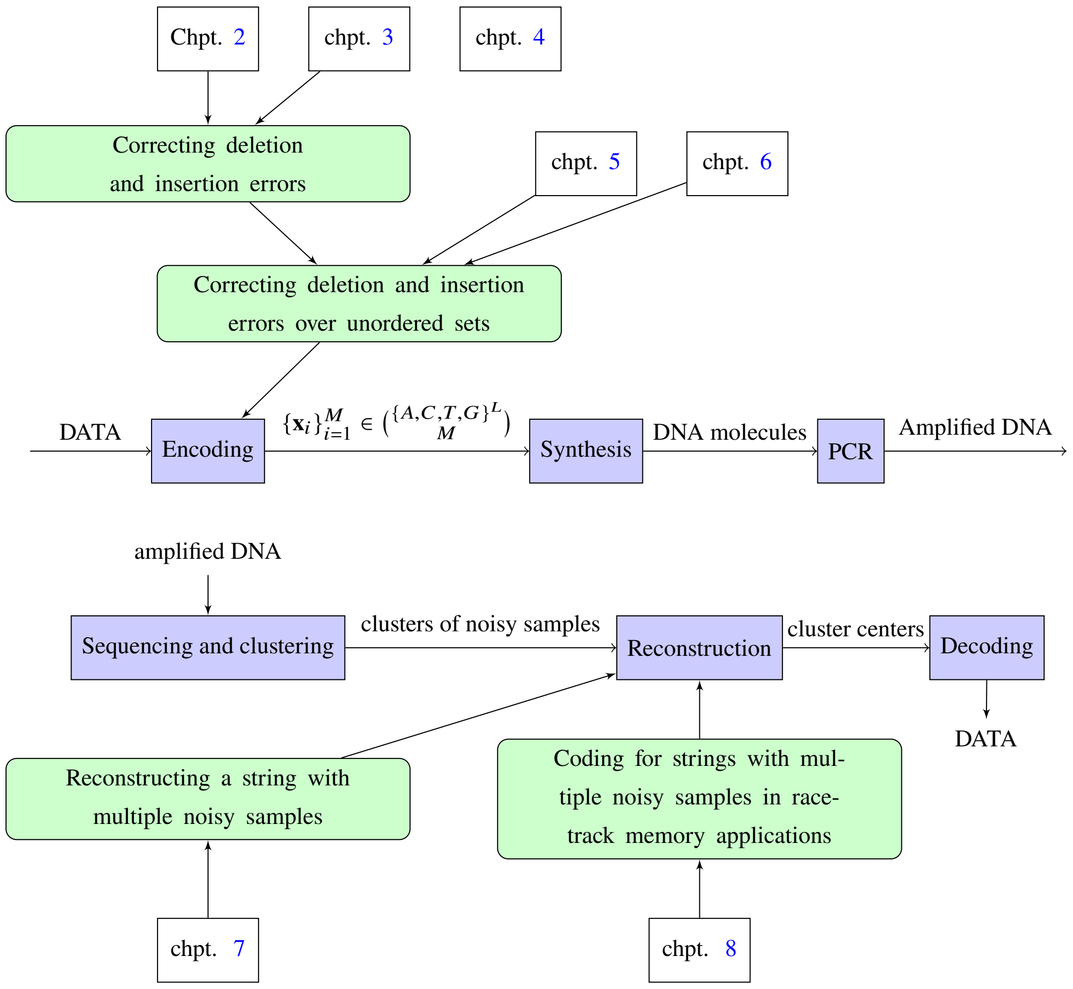
Chpt. 2
chpt. 3
chpt. 4
Correcting deletion and insertion errors
chpt. 5
chpt. 6
Correcting deletion and insertion errors over unordered sets
DATA
Encoding
$\{\mathbf{x}_i\}_{i=1}^M \in \binom{\{A,C,T,G\}^L}{M}$
Synthesis
DNA molecules
PCR
Amplified DNA
amplified DNA
Sequencing and clustering
clusters of noisy samples
Reconstruction
cluster centers
Decoding
DATA
Reconstructing a string with multiple noisy samples
Coding for strings with multiple noisy samples in race-track memory applications
chpt. 7
chpt. 8

Figure 1.2: Contribution and organization of this thesis

1.3 Contributions

Fig. 1.2 illustrates our contributions and the organization of this thesis, in the context of the workflow of DNA-based storage systems, described in Sec. 1.1. We address the aforementioned three challenges for error correction in DNA-based storage, which are described in Sec. 1.2 and arise in encoding/decoding procedures and the reconstruction procedure in DNA storage systems. The optimal error correcting solution for DNA storage is a joint encoder and decoder design, which takes into account all three challenges. However, as each challenge itself is a hard task, we address them separately, hoping that our techniques provide building blocks for joint encoder decoder design. Interestingly, our results for the first and second challenges together provide an efficient solution for correcting deletion/insertion errors under some settings. Furthermore, our result for the first challenge was used for dealing with the third challenge, which looks different and independent from the first challenge. We also investigate how error correction codes can be used for the third challenge, though for different applications. In the following we summarize our contributions and give an outline of this thesis.

Ch. 2, Ch. 3, and Ch. 4 address the first challenge, constructing codes correcting deletion and insertion errors. The problem of constructing efficient codes correcting a constant number of deletions/insertions has long been unsettled even for two deletions. In Ch. 2, we start from an efficient binary 2-deletion correcting code construction, which generalizes a classic code construction correcting a single deletion error. Then, in Ch. 3 we extend it to correct any constant number of deletion/insertions. Our construction achieves redundancy at most eight times the optimal. In Ch. 4, we further develop the techniques in Ch. 3, and proposed a systematic binary deletion/insertion correcting codes with redundancy at most four times the optimal. Asymptotically, our code construction is the current best known. We then extend our results and proposed deletion/insertion error correcting codes under various settings, including non-binary deletion/insertion correcting codes, systematic binary codes correcting a burst of variable length deletions, codes correcting a constant number of burst deletions, each occur within a window of limited length. All of our constructions achieve order-wise optimal redundancy.

In Ch. 5 and Ch. 6, we deal with the second challenge and propose codes over an unordered set of strings that correct deletion/insertion and substitution errors. In Ch. 5, we consider substitution errors only and derive upper and lower bounds on the redundancy of error correcting codes over sets. We find a surprising result that the redundancy needed for correcting a set of unordered strings is equal to that needed for correcting a set of strings given their order. We also provide an efficient code construction, that outperforms the state-of-the-art under some settings. In Ch. 6, we further improve our construction in Ch. 5, and present a code correcting deletion/insertion and substitution errors with order-wise optimal redundancy. This generalizes our results in Ch. 3 and Ch. 4, where the set has a single string. One of our techniques, which we refer to as *robust indexing*, uses data to index themselves. The robust indexing idea provides an alternative approach to currently used index-based schemes.

Ch. 7 is about the third challenge. We aim to show that a string can be reconstructed given a polynomial number of its noisy copies. As the problem is difficult and open in general, we take a less ambitious goal and show that a polynomial number of noisy copies suffices to reconstruct the string, given a reference string that is within constant edit distance from the target string. The edit distance measures the similarity between two strings. Our result might have applications in the Human Genome Project, where a human reference genome is provided to study the difference between individual genomes.

Finally, Ch. 8 studies how error correction codes help reconstruct a string from its multiple noisy copies. Instead of looking for applications in DNA storage, we investigate multi-head racetrack memory applications, which have a different model on noisy copies from the one in Ch. 7. Racetrack memory is an emerging storage technique that has the potential of high storage density (not as high as DNA storage theoretically can be) and low latency. In multi-head racetrack memory, each head produces a noisy copy of the string. Different from Ch. 7, where noisy copies are independently and randomly generated, in this chapter, the noisy copies are correlated. We provide codes that correct a combination of any constant number of deletions and insertions, with order-wise optimal efficiency. The techniques we use in this chapter might in turn inspire new ideas to deal with the trace reconstruction problem in Ch. 7, which we leave for future explorations.

Chapter 2

BINARY 2-DELETION/INSERTION CORRECTING CODES

To study correction of deletion/insertion errors from its most basic form, we begin with binary cases where the codewords are sequences of bits. In this chapter, we present codes correcting two deletions and/or insertions. While deletion correcting codes for more general cases will be given in the next chapters, the construction in this chapter contains some of the original ideas that we develop for general settings, including a generalization of the VT codes and computing parity checks of a sequence from symbols in its indicator vectors.

2.1 Introduction

A deletion in a binary sequence $\boldsymbol{c} = (c_1, \ldots, c_n) \in \{0,1\}^n$ is the case where a symbol is removed from $\boldsymbol{c}$, which results in a subsequence length $n-1$. Similarly, the result of a k-deletion is a subsequence of $\boldsymbol{c}$ of length $n-k$. A k-deletion code C is a set of n-bit sequences, no two of which share a common subsequence of length $n-k$; and clearly, such a code can correct any k-deletion.

The problem of constructing a k-deletion correcting code was introduced by Levenshtein [64], who showed the equivalence between correcting k deletions and correcting r deletions and s insertions whenever $r+s \leq k$, for a single sequence. Therefore, to correct deletion and insertion errors in a single sequence, we only need to consider deletion correcting codes. Levenshtein proved that the optimal redundancy (defined as $n - \log |C|$) is $O(k \log n)$ for constant k. Specifically, it is in the range $k \log n + o(\log n)$ to $2k \log n + o(\log n)$ when k is a constant. In general, the redundancy $O(k \log n)$ is order-wise optimal when $k \leq O(n^\epsilon)$ for some $\epsilon < 1$. The optimal redundancy becomes $O(k \log(n/k))$ when $k = O(n)$ and k is small, e.g., $k = n/4$ [22]. When $k \geq n/2$, a k-deletion correcting code has cardinality at most two. In addition, Levenshtein proposed the following optimal construction (the well-known Varshamov-Tenengolts (VT) code [95]):

$$\left\{(c_1, \ldots, c_N) : \sum_{i=1}^{N} i c_i \equiv 0 \bmod (n+1)\right\}, \tag{2.1}$$

that is capable of correcting a single deletion with redundancy not more than $\log(n+1)$ [64]. The encoding/decoding complexity of VT codes is linear in n. Generalizing the VT construction to correct more than a single deletion was elusive for more than 50 years. In particular, the past approaches [47], [3], [74] resulted in asymptotic code rates that were bounded away from 1.

A breakthrough paper [12] proposed a k-deletion correcting code construction with $O(k^2 \log k \log n)$ redundancy and $O_k(n \log^4 n)$[1] encoding/decoding complexity, where k is a constant. This code is based on a k-deletion code of length $\log n$, which is constructed using a computer search. Nevertheless, the constants that are involved in the work of [12] are orders of magnitude away from the lower bound in $\log n + o(\log n)$ even for $k = 2$, and the code is not systematic.

For $k = 2$, [34] improved the redundancy up to $8 \log n + o(\log n)$ using techniques similar to [12], which are fundamentally different from ours, and incur higher redundancy and complexity. Moreover, finding a k-deletion correcting code with an asymptotic rate 1 as an extension of the VT construction remained widely open. More recently, a 2-deletion correcting code construction with redundancy $4 \log n + o(\log n)$ was proposed in [38], which improved our $7 \log n + o(\log n)$ bits of redundancy. The code construction in [38] is not systematic and used a different approach from ours, which is hard to generalize to correct more deletions.

In this chapter, we provide a systematic two-deletion correcting code with redundancy $7 \log n + o(\log n)$ and linear encoding/decoding complexity. The main result is as follows.

Theorem 2.1.1. *For any integer $n \geq 3$ and $N = n + 7 \log n + o(\log n)$, there exists an encoding function $\mathcal{E} : \{0,1\}^n \to \{0,1\}^N$ and a decoding function $\mathcal{D} : \{0,1\}^{N-2} \to \{0,1\}^n$ such that for any $\mathbf{c} \in \{0,1\}^n$ and subsequence $\mathbf{c}' \in \{0,1\}^{N-2}$ of $\mathcal{E}(\mathbf{c})$, we have that $\mathcal{D}(\mathbf{c}') = \mathbf{c}$. In addition, functions $\mathcal{E}$ and $\mathcal{D}$ can be computed in $O(n)$ time.*

The encoding and decoding functions $\mathcal{E}$ and $\mathcal{D}$ will be explicitly constructed, based on a VT-like extension that will be presented next. One might conjecture that a potential extension of the VT code can be obtained by using higher order parity checks $\sum_{i=1}^{n} i^j c_i \bmod (n^j + 1)$ for $j = 0, 1, \ldots, t$, but counterexamples are constructible even for $k = 2$ and $t \leq 4$ [12]. The following is a counterexample for $k = 2$ and $t = 3$. Let

$$\mathbf{c} = (1,0,0,0,1,0,1,1,1,0,0,1) \text{ and}$$

[1]The notion O_k denotes *parameterized complexity*, i.e., $O_k(n \log^4 n) = f(k)O(n \log^4 n)$ for some function f.

$$\mathbf{c}' = (0,1,1,0,0,0,1,0,1,1,1,0)$$

be two sequences of length 12. The sequences $\mathbf{c}$ and $\mathbf{c}'$ share a common subsequence $(1,0,0,0,1,0,1,1,1,0)$ of length 10. It can be verified that $\sum_{i=1}^{n} i^j c_i = \sum_{i=1}^{n} i^j c'_i$ for $j \in \{0,1,2,3\}$. It is not known whether there is a constant bound t such that the higher order parity check works for two deletions.

In this section, we find that similar higher order parity checks work when $t = 3$, given that we restrict our attention to sequences with no adjacent ones. In particular, for sequences with no adjacent ones, the "syndrome," i.e., the difference between the parity checks of a codeword and an erroneous sequence, can be expressed as a linear function. The matrix of the linear function is similar to the Vandermonde matrix in the sense that its columns are the sums of Vandermonde matrix columns. It can be shown that such a matrix has a positive determinant and hence the "syndrome" cannot be zero. Note that in the above counterexample both $\mathbf{c}$ and $\mathbf{c}'$ contain adjacent ones. Consequently, applying these parity checks on certain *indicator vectors* yields the desired result. For a and b in $\{0,1\}$ and a binary sequence $\boldsymbol{c}$, the ab-indicator $\mathbb{1}_{ab}(\boldsymbol{c}) \in \{0,1\}^{n-1}$ of $\boldsymbol{c}$ is

$$\mathbb{1}_{ab}(c)_i = \begin{cases} 1 & \text{if } c_i = a \text{ and } c_{i+1} = b \\ 0 & \text{else.} \end{cases}.$$

For example,

$$\begin{aligned} \mathbf{c} &= (1,0,1,0,0,1,1,0,0,1,1,0,0,0) \\ \mathbb{1}_{10}(\mathbf{c}) &= (1,0,1,0,0,0,1,0,0,0,1,0,0) \\ \mathbb{1}_{01}(\mathbf{c}) &= (0,1,0,0,1,0,0,0,1,0,0,0,0). \end{aligned}$$

Since any two 10 or 01 patterns are at least two positions apart, the 10- and 01-indicators of any n-bit sequence do not contain consecutive ones, and hence higher order parity checks can be applied. The parity checks in the proposed code rely on the following integer vectors.

$$\begin{aligned} \mathbf{m}^{(0)} &\triangleq (1,2,\ldots,n-1) \\ \mathbf{m}^{(1)} &\triangleq \left(1, 1+2, 1+2+3, \ldots, \frac{n(n-1)}{2}\right) \\ \mathbf{m}^{(2)} &\triangleq \left(1^2, 1^2+2^2, 1^2+2^2+3^2, \ldots, \frac{n(n-1)(2n-1)}{6}\right), \end{aligned}$$

and further, for $\mathbf{c} \in \{0,1\}^n$ let

$$\begin{aligned} f(\mathbf{c}) \triangleq (&\mathbb{1}_{10}(\mathbf{c}) \cdot \mathbf{m}^{(0)} \bmod 2n, \\ &\mathbb{1}_{10}(\mathbf{c}) \cdot \mathbf{m}^{(1)} \bmod n^2, \\ &\mathbb{1}_{10}(\mathbf{c}) \cdot \mathbf{m}^{(2)} \bmod n^3), \text{ and} \\ h(\mathbf{c}) \triangleq (&\mathbb{1}_{01}(\mathbf{c}) \cdot \mathbb{1} \bmod 3, \mathbb{1}_{01}(\mathbf{c}) \cdot \mathbf{m}^{(1)} \bmod 2n), \end{aligned} \tag{2.2}$$

where $\cdot$ denotes inner product over the integers, and $\mathbb{1}$ denotes the all 1's vector. The term $\mathbb{1}_{01}(\mathbf{c}) \cdot \mathbb{1}$ in (2.2) counts the number of 1-runs that are preceded by a 0 in the vector $\mathbb{1}_{01}(\mathbf{c})$. The functions $f(\mathbf{c})$ and $h(\mathbf{c})$ are used to obtain parity symbols in our construction. To show this, we prove the following theorem, which requires the notion of a *k-deletion ball*. For any integer k let $B_k(\mathbf{c})$ be the k-deletion ball of $\mathbf{c}$, i.e., the set of n-bit sequences that share a common length $n-k$ subsequence with $\mathbf{c}$.

Theorem 2.1.2. *For $\mathbf{c}, \mathbf{c}' \in \{0,1\}^n$, if $\mathbf{c} \in B_2(\mathbf{c}')$, $f(\mathbf{c}) = f(\mathbf{c}')$, and $h(\mathbf{c}) = h(\mathbf{c}')$, then $\mathbf{c} = \mathbf{c}'$.*

Theorem 2.1.2 indicates the possibility of constructing a 2-deletion code, with functions $h(\mathbf{c})$ and $f(\mathbf{c})$ serving as the redundancy bits. The induced redundancy is at most $7\log(n)+o(\log n)$ (the $o(\log n)$ term stems from both the redundancy $h(\mathbf{c})$ and $f(\mathbf{c})$ and the extra redundancy to protect $h(\mathbf{c})$ and $f(\mathbf{c})$). Furthermore, the encoding algorithm is straightforward — we begin by appending the redundancies $f(\mathbf{c})$ and $h(\mathbf{c})$ to the sequence $\mathbf{c}$, in their binary representation. Then, to protect the redundancies $f(\mathbf{c})$ and $h(\mathbf{c})$, we apply functions $f(\cdot)$ and $h(\cdot)$ once again and append $f(f(\mathbf{c}), h(\mathbf{c}))$ and $h(f(\mathbf{c}), h(\mathbf{c}))$ to the sequences[2]. Finally, in order to protect the latter, we use a simple 3-fold repetition code. The encoding function is given in the following construction.

Construction 2.1.1. *For a sequence $\mathbf{c} \in \{0,1\}^n$, the encoding function is*

$$\mathcal{E}(\mathbf{c}) = (\mathbf{c}, f(\mathbf{c}), h(\mathbf{c}), r_3(f(f(\mathbf{c}), h(\mathbf{c}))), r_3(h(f(\mathbf{c}), h(\mathbf{c})))), \tag{2.3}$$

where r_3 is a 3-fold repetition encoding function. The length of the codeword $\mathcal{E}(\mathbf{c})$ is

$$\begin{aligned} N \triangleq& n + 7\log n + 8 + 21\log(7\log n + 8) + 24 \\ =& n + 7\log n + o(\log n). \end{aligned}$$

[2]Note that the functions $f(\cdot)$ and $h(\cdot)$ are applied here on the sequences $\mathbf{c}$, and later on the sequence $(f(\mathbf{c}), h(\mathbf{c}))$, that is shorter than $\mathbf{c}$. However, it readily follows from (2.2) that functions $f(\cdot)$ and $h(\cdot)$ can be defined over sequences of any length.

Clearly, the computation of the function $\mathcal{E}(\mathbf{c})$ can be done in linear time. The linear time decoding of Construction 2.1.1 is achieved by a reduction to the problem of finding an element in a sorted matrix of integers; a problem which is solvable in linear time. In addition, we show that this matrix does not have to be calculated in full, since each entry can be computed from its neighbors in constant time. The decoding algorithm will be given in detail in Sec. 2.5. Construction 2.1.1 and the linear decoding algorithm together prove Theorem 2.1.1.

One of the most substantial aspects of our contribution lies in viewing it as a generalization of the VT construction. In particular, the proof of Theorem 2.1.2 can be seen as a higher dimensional variant of the proof for the VT construction. In the remainder of this section, a generalized proof of correctness for the VT codes is presented. Then, it is shown how this particular proof can be extended to prove Theorem 2.1.2. To the best of the authors' knowledge, this constitutes the first extension of the VT code which attains rate 1.

Generalizing VT for a single deletion

Clearly, a VT code of length $n-1$ can be seen as a set of sequences $\mathbf{c}$ for which the values of $\ell(\mathbf{c}) \triangleq \mathbf{c}\cdot\mathbf{m}^{(0)} \bmod n$ coincide. Adopting this point of view, the correctness of the VT construction is an immediate corollary of the following more general claim, in which $\ell_{\mathbf{v}}(\mathbf{c}) \triangleq \mathbf{c}\cdot\mathbf{v} \bmod (v_{n-1}+1)$. The claim also appeared in [43]. Here we prove it in a different approach that will be used throughout the chapter.

Proposition 2.1.1. *For $\mathbf{c},\mathbf{c}' \in \{0,1\}^{n-1}$, and $\mathbf{v}\in\mathbb{Z}_+^{n-1}$, if $\mathbf{c}\in B_1(\mathbf{c}')$, $\ell_{\mathbf{v}}(\mathbf{c}) = \ell_{\mathbf{v}}(\mathbf{c}')$, and $v_1 < v_2 < \ldots < v_{n-1}$ then $\mathbf{c}=\mathbf{c}'$.*

In turn, the proof of this proposition can be completed by defining the following function. For a vector $\mathbf{v}\in\mathbb{Z}_+^{n-1}$, an integer $r\in\{1,\ldots,n-1\}$, and a binary vector $\mathbf{x}=(x_1,\ldots,x_s)$ with $r+s-2\le n-1$, let

$$\begin{aligned} g_{\mathbf{v}}(r,\mathbf{x}) &\triangleq \mathbf{x}\cdot((\mathbf{v}^{(r,r+s-2)},0)-(0,\mathbf{v}^{(r,r+s-2)})) \\ &= x_1 v_r - x_s v_{r+s-2} + \sum_{t=2}^{s-1} x_t(v_{t+r-1}-v_{t+r-2}), \end{aligned} \tag{2.4}$$

where $\mathbf{v}^{(r,r+s-2)} \triangleq (v_r, v_{r+1},\ldots,v_{r+s-2})$. As shown in the proof of Proposition 2.1.1, the difference $\ell_{\mathbf{v}}(\mathbf{c})-\ell_{\mathbf{v}}(\mathbf{c}')$ for $\mathbf{c}\in B_1(\mathbf{c}')$, which plays a key role in proving Proposition 2.1.1, can be expressed in the form $g_{\mathbf{v}}(r,\mathbf{x})$ for some x. Furthermore, the difference of parity checks $f(\mathbf{c}) - f(\mathbf{c}')$ for $\mathbf{c} \in B_2(\mathbf{c}')$ can be expressed as the sum or difference of two functions $g_{\mathbf{v}}(r_1, \mathbf{x}_1)$ and $g_{\mathbf{v}}(r_2, \mathbf{x}_2)$. The following claim regarding the function $g_{\mathbf{v}}(r, \mathbf{x})$ will be used to prove Proposition 2.1.1.

Proposition 2.1.2. *For positive integers r and s such that $r + s - 2 \le n - 1$, a vector $\mathbf{v} \in \mathbb{Z}_+^{n-1}$ with $v_1 < \ldots < v_{n-1}$, and an s-bit binary vector $\mathbf{x}$, if $g_{\mathbf{v}}(r, \mathbf{x}) = 0$, then $\mathbf{x}$ is a constant vector.*

Proof. We distinguish between two cases according to the value of x_s. On the one hand, if $x_s = 0$, then it is readily verified according to (2.4) that $g_{\mathbf{v}}(r, \mathbf{x})$ is the sum of nonnegative terms. In which case, the equation $g_{\mathbf{v}}(r, \mathbf{x}) = 0$ holds if and only if $\mathbf{x} = 0$. On the other hand, if $x_s = 1$, then

$$\begin{aligned} g_{\mathbf{v}}(r, \mathbf{x}) &= v_r x_1 + \sum_{t=2}^{s-1} (v_{t+r-1} - v_{t+r-2}) x_t - v_{r+s-2} \\ &\le v_r + \sum_{t=2}^{s-1} (v_{t+r-1} - v_{t+r-2}) - v_{r+s-2} = 0. \end{aligned} \tag{2.5}$$

The equality holds if and only if $\mathbf{x} = 1$. $\square$

Proof. (of Proposition 2.1.1) Let k_1 and k_2 ($k_1 \le k_2$) be the indices of the deletions in $\mathbf{c}$ and $\mathbf{c}'$, respectively, after which they are identical. Then, we have

$$\begin{aligned} \mathbf{c}_t &= \mathbf{c}'_t && \text{if } t < k_1 \\ & && \text{or } t > k_2, \text{ and} \\ \mathbf{c}_{t+1} &= \mathbf{c}'_t && \text{if } k_1 \le t \le k_2 - 1, \end{aligned} \tag{2.6}$$

and one can find that

$$\begin{aligned} \mathbf{c} \cdot \mathbf{v} - \mathbf{c}' \cdot \mathbf{v} = & \sum_{t=1}^{k_1-1} c_t v_t + c_{k_1} v_{k_1} + \sum_{t=k_1+1}^{k_2} c_t v_t + \\ & \sum_{t=k_2+1}^{n} c_t v_t - (\sum_{t=1}^{k_1-1} c_t v_t + \sum_{t=k_1}^{k_2-1} c_{t+1} v_t + \\ & c'_{k_2} v_{k_2} + \sum_{t=k_2+1}^{n} c_t v_t) \\ & = c_{k_1} v_{k_1} + \sum_{t=k_1+1}^{k_2} c_t (v_t - v_{t-1}) - c_{k_2} v_{k-2} \\ & = g_{\mathbf{v}}(k_1, (\mathbf{c}^{(k_1,k_2)}, c'_{k_2})). \end{aligned} \tag{2.7}$$

Hence, if $\ell_v(\mathbf{c}) = \ell_v(\mathbf{c}')$, then we have that $g_{\mathbf{v}}(k_1, (\mathbf{c}^{(k_1,k_2)}, c'_{k_2})) \equiv 0 \bmod (v_{n-1}+1)$. Furthermore, since

$$\begin{aligned} -v_{n-1} &\le -v_{r+s-2} \\ &\le x_1 v_r - x_s v_{r+s-2} + \sum_{t=2}^{s-1} x_t (v_{t+r-1} - v_{t+r-2}) \\ &\le v_r + \sum_{t=2}^{s-1} (v_{t+r-1} - v_{t+r-2}) = v_{r+s-2} \le v_{n-1}, \end{aligned} \tag{2.8}$$

it follows that $\ell_{\mathbf{v}}(\mathbf{c}) = \ell_{\mathbf{v}}(\mathbf{c}')$ if and only if $g_{\mathbf{v}}(k_1, (\mathbf{c}^{(k_1,k_2)}, c'_{k_2})) = 0$. The proof is now concluded by using Proposition 2.1.2 and Eq. (2.6) as follows. From Proposition 2.1.2 we get $c_{k_1} = \ldots = c_{k_2} = c'_{k_2}$. Together with Eq. (2.6), we have that $c'_t = c_{t+1} = c_t$ for $k_1 \le t \le k_2 - 1$ and $c'_t = c_t$ for $t < k_1$ or $t \ge k_2$, and hence $\mathbf{c} = \mathbf{c}'$. □

The proof of correctness for a VT code of length $n-1$ immediately follows from Proposition 2.1.1 by choosing $\mathbf{v} = (1, \ldots, n-1)$. Now, the crux of proving Theorem 2.1.2 boils down to the following higher dimensional variant of Proposition 2.1.2, and hence the tight connection between the VT construction and the one which is presented in this chapter.

Proposition 2.1.3. *For positive integers r_1, r_2, s_1, and s_2 such that $r_2 > r_1 + s_1 - 2$ and $r_2 + s_2 - 2 \le n - 1$, and binary sequences $\mathbf{x}$ and $\mathbf{y}$ of lengths s_1 and s_2, respectively, if*

$$\begin{aligned} g_{\mathbf{m}^{(0)}}(r_1, \mathbf{x}) + \lambda g_{\mathbf{m}^{(0)}}(r_2, \mathbf{y}) &= 0, \text{ and} \\ g_{\mathbf{m}^{(1)}}(r_1, \mathbf{x}) + \lambda g_{\mathbf{m}^{(1)}}(r_2, \mathbf{y}) &= 0, \end{aligned} \tag{2.9}$$

where $\lambda = \pm 1$, then $\mathbf{x}$ and $\mathbf{y}$ are constant vectors.

Additional technical claims, which involve the remaining ingredients of the redundancy bits, are given in the sequel.

2.2 Outline

The intuition behind applying the redundancy functions over the indicator vectors, rather than over the message itself, is the following simple lemma.

Lemma 2.2.1. *For $\mathbf{c}$ and $\mathbf{c}'$ in $\{0,1\}^n$ such that $\mathbf{c} \in B_2(\mathbf{c}')$, if $\mathbb{1}_{10}(\mathbf{c}) = \mathbb{1}_{10}(\mathbf{c}')$ and $\mathbb{1}_{01}(\mathbf{c}) = \mathbb{1}_{01}(\mathbf{c}')$ then $\mathbf{c} = \mathbf{c}'$.*

Proof. The conditions $\mathbb{1}_{10}(\boldsymbol{c}) = \mathbb{1}_{10}(\boldsymbol{c}')$ and $\mathbb{1}_{01}(\boldsymbol{c}) = \mathbb{1}_{01}(\boldsymbol{c}')$ imply that the ascending (i.e., 0 to 1) and descending (i.e., 1 to 0) transition positions of $\boldsymbol{c}$ coincide with those of $\boldsymbol{c}'$ respectively. Hence if transitions exist in $\boldsymbol{c}$ or $\boldsymbol{c}'$, then $\boldsymbol{c} = \boldsymbol{c}'$. If no transitions happen in $\boldsymbol{c}$ or $\boldsymbol{c}'$ and $\boldsymbol{c} \neq \boldsymbol{c}'$, then one of $\boldsymbol{c}$ and $\boldsymbol{c}'$ is all 0's vector and the other is all 1's vector. Since all 0's vector does not share a common subsequence of length $n-2$ with all 1's vector, the claim follows. □

Based on this lemma, the proof of Theorem 2.1.2 is separated to the following two lemmas. Generally speaking, it is shown that for two confusable sequences, i.e., that share a common $n-2$ subsequence, if the f redundancies coincide (2.2), then so are the 10-indicators. Then, it is shown that confusable sequences with identical 10-indicators and identical h redundancy, have identical 01-indicators.

Lemma 2.2.2. *For* $\mathbf{c}$ *and* $\mathbf{c}'$ *in* $\{0,1\}^n$*, if* $\mathbf{c} \in B_2(\mathbf{c}')$ *and* $f(\mathbf{c}) = f(\mathbf{c}')$*, then* $\mathbb{1}_{10}(\mathbf{c}) = \mathbb{1}_{10}(\mathbf{c}')$.

Lemma 2.2.3. *For* $\mathbf{c}$ *and* $\mathbf{c}'$ *in* $\{0,1\}^n$ *such that* $\mathbf{c} \in B_2(\mathbf{c}')$*, if* $\mathbb{1}_{10}(\mathbf{c}) = \mathbb{1}_{10}(\mathbf{c}')$ *and* $h(\mathbf{c}) = h(\mathbf{c}')$*, then* $\mathbb{1}_{01}(\mathbf{c}) = \mathbb{1}_{01}(\mathbf{c}')$.

From these lemmas it is clear that two n-bit sequences that share a common $n-2$ subsequence and agree on the redundancies f and h have identical 10- and 01-indicators, and hence, the proof of Theorem 2.1.2 is concluded. The proofs of Lemma 2.2.2 and Lemma 2.2.3 make extensive use of the following two technical claims, that are easy to prove.

Proposition 2.2.1. *For* $\mathbf{c}$ *and* $\mathbf{c}'$ *in* $\{0,1\}^n$*, if* $\mathbf{c} \in B_2(\mathbf{c}')$ *then* $\mathbb{1}_{10}(\mathbf{c}) \in B_2(\mathbb{1}_{10}(\mathbf{c}'))$ *and* $\mathbb{1}_{01}(\mathbf{c}) \in B_2(\mathbb{1}_{01}(\mathbf{c}'))$.

Proof. We first show that if $\boldsymbol{c} \in B_1(\boldsymbol{c}')$ then $\mathbb{1}_{10}(\boldsymbol{c}) \in B_1(\mathbb{1}_{10}(\boldsymbol{c}'))$ and $\mathbb{1}_{01}(\boldsymbol{c}) \in B_1(\mathbb{1}_{01}(\boldsymbol{c}'))$. To this end, it suffices to show that if $\mathbf{d} \in \{0,1\}^{n-1}$ is obtained from $\boldsymbol{c}$ by one deletion, then $\mathbb{1}_{10}(\mathbf{d})$ (resp. $\mathbb{1}_{01}(\mathbf{d})$) is obtained from $\mathbb{1}_{10}(\boldsymbol{c})$ (resp. $\mathbb{1}_{01}(\boldsymbol{c})$) by one deletion (see Table 2.1). Further, it is easy to see that a deletion of c_1 corresponds to a deletion of $\mathbb{1}_{10}(\mathbf{c})_1$ (resp. $\mathbb{1}_{01}(\mathbf{c})_1$) and a deletion of c_n corresponds to a deletion of $\mathbb{1}_{10}(\mathbf{c})_{n-1}$ (resp. $\mathbb{1}_{01}(\mathbf{c})_{n-1}$). Hence, it follows that if

$$\mathbf{c} \xrightarrow{\text{1 del'}} \mathbf{d} \xrightarrow{\text{1 del'}} \mathbf{e}$$

$$\mathbf{c}' \xrightarrow{\text{1 del'}} \mathbf{d}' \xrightarrow{\text{1 del'}} \mathbf{e}$$

Table 2.1: All possible cases of deletions of c_i for $2 \leq i \leq n-1$ correspond to deletions in $\mathbb{1}_{10}(\boldsymbol{c})$ (resp. $\mathbb{1}_{01}(\mathbf{c}')$).

$c_{i-1}c_ic_{i+1}$	000	001	010	011	100	101	110	111
$\mathbb{1}_{10}(c)_{i-1}\mathbb{1}_{10}(c)_i$	**00**	**0**0	0**1**	**0**0	1**0**	**1**0	**01**	**00**
$\mathbb{1}_{01}(c)_{i-1}\mathbb{1}_{01}(c)_i$	**00**	0**1**	**10**	**1**0	**00**	**0**1	**0**0	**00**

then

$$\mathbb{1}_{10}(\mathbf{c}) \overset{1\text{ del'}}{\longrightarrow} \mathbb{1}_{10}(\mathbf{d}) \overset{1\text{ del'}}{\longrightarrow} \mathbb{1}_{10}(\mathbf{e})$$
$$\mathbb{1}_{10}(\mathbf{c}') \overset{1\text{ del'}}{\longrightarrow} \mathbb{1}_{10}(\mathbf{d}') \overset{1\text{ del'}}{\longrightarrow} \mathbb{1}_{10}(\mathbf{e})$$
$$\mathbb{1}_{01}(\mathbf{c}) \overset{1\text{ del'}}{\longrightarrow} \mathbb{1}_{01}(\mathbf{d}) \overset{1\text{ del'}}{\longrightarrow} \mathbb{1}_{01}(\mathbf{e})$$
$$\mathbb{1}_{01}(\mathbf{c}') \overset{1\text{ del'}}{\longrightarrow} \mathbb{1}_{01}(\mathbf{d}') \overset{1\text{ del'}}{\longrightarrow} \mathbb{1}_{01}(\mathbf{e}),$$

which concludes the claim. □

Proposition 2.2.2. *For* $\mathbf{c}, \mathbf{c}' \in \{0,1\}^n$, *if* $\mathbf{c} \in B_2(\mathbf{c}')$ *and* $\mathbb{1}_{01}(\mathbf{c})\cdot\mathbb{1} = \mathbb{1}_{01}(\mathbf{c}')\cdot\mathbb{1} \bmod 3$, *then* $\mathbb{1}_{01}(\mathbf{c}) \cdot \mathbb{1} = \mathbb{1}_{01}(\mathbf{c}') \cdot \mathbb{1}$.

Proof. Since $\boldsymbol{c} \in B_2(\boldsymbol{c}')$, it follows from Proposition 2.2.1 that $\mathbb{1}_{10}(\boldsymbol{c}) \in B_2(\mathbb{1}_{10}(\boldsymbol{c}'))$, and thus $\mathbb{1}_{10}(\boldsymbol{c})$ and $\mathbb{1}_{10}(\boldsymbol{c}')$ have a mutual $(n-3)$-bit substring $\mathbf{s}$. Since $\mathbf{s}\cdot\mathbb{1}$, $\mathbb{1}_{10}(\boldsymbol{c})\cdot\mathbb{1}$, and $\mathbb{1}_{10}(\boldsymbol{c}')\cdot\mathbb{1}$ are the number of 1 entries in $\mathbf{s}$, $\mathbb{1}_{10}(\boldsymbol{c})$, and $\mathbb{1}_{10}(\boldsymbol{c}')$ respectively, we have that

$$\mathbf{s}\cdot\mathbb{1} \leq \mathbb{1}_{10}(\boldsymbol{c})\cdot\mathbb{1} \leq \mathbf{s}\cdot\mathbb{1} + 2, \text{ and}$$
$$\mathbf{s}\cdot\mathbb{1} \leq \mathbb{1}_{10}(\boldsymbol{c}')\cdot\mathbb{1} \leq \mathbf{s}\cdot\mathbb{1} + 2,$$

and thus $|\mathbb{1}_{10}(\boldsymbol{c})\cdot\mathbb{1} - \mathbb{1}_{10}(\boldsymbol{c}')\cdot\mathbb{1}| \leq 2$. However, since 3 divides $|\mathbb{1}_{10}(\boldsymbol{c})\cdot\mathbb{1} - \mathbb{1}_{10}(\boldsymbol{c}')\cdot\mathbb{1}|$, we must have that $\mathbb{1}_{10}(\boldsymbol{c})\cdot\mathbb{1} = \mathbb{1}_{10}(\boldsymbol{c}')\cdot\mathbb{1}$. □

In addition, one of the cases of the proof of Lemma 2.2.2 requires a specialized variant of Proposition 2.1.3, as follows.

Proposition 2.2.3. *Let* r_1, r_2, s_1, s_2 *and* s_3 *be positive integers that satisfy* $r_2 = r_1 + s_1$ *and* $r_2 + s_2 + s_3 - 1 \leq n - 1$, *and let* $\mathbf{x} \in \{0,1\}^{s_1+s_2+1}$ *and* $\mathbf{y} \in \{0,1\}^{1+s_2+s_3}$ *be such that*

$$(x_{s_1+1}, x_{s_1+2}, \ldots, x_{s_1+s_2}) = (y_2, y_3, \ldots, y_{s_2+1}),$$

and $(x_{s_1+1}, x_{s_1+2}, \ldots, x_{s_1+s_2})$ *has no adjacent* 1*'s. If*

$$\begin{aligned} &g_{\mathbf{m}^{(0)}}(r_1, \mathbf{x}) + g_{\mathbf{m}^{(0)}}(r_2, \mathbf{y}) = 0, \\ &g_{\mathbf{m}^{(1)}}(r_1, \mathbf{x}) + g_{\mathbf{m}^{(1)}}(r_2, \mathbf{y}) = 0, \text{ and} \\ &g_{\mathbf{m}^{(2)}}(r_1, \mathbf{x}) + g_{\mathbf{m}^{(2)}}(r_2, \mathbf{y}) = 0, \end{aligned} \tag{2.10}$$

then either $x_1 = \ldots = x_{s_1+s_2+1} = y_1 = \ldots = y_{s_2+s_3+1}$ *or*

$$\begin{aligned} &x_1 = x_2 = \ldots = x_{s_1+1} = 1 - y_1, \\ &x_t + x_{t+1} = 1, \text{ for } t \in \{s_1+1, \ldots, s_1+s_2-1\}, \\ &x_{s_1+s_2+1} + y_{s_2+1} = 1, \text{ and} \\ &y_{s_2+1} = \ldots = y_{s_2+s_3+1}. \end{aligned} \tag{2.11}$$

Proposition 2.1.3 and Proposition 2.2.3 are key to the proof of Lemma 2.2.2. Moreover, the proof of Proposition 2.2.3 contains one of the main ideas in this chapter that proves the correctness of correcting two deletions using higher order parity checks. Proposition 2.1.3 and Proposition 2.2.3 are proved in Sec. 2.3. Finally, the following lemma shows a property of $g_{\mathbf{v}}(r, \mathbf{x})$, which will be useful in the proof of Proposition 2.1.3 and Proposition 2.2.3.

Proposition 2.2.4. *For positive integers r and s such that $r + s - 2 \leq n - 1$, a vector* $\mathbf{v}$*, and an s-bit binary vector* $\mathbf{x}$*, we have that*

$$g_{\mathbf{v}}(r, \mathbf{x}) + g_{\mathbf{v}}(r, \overline{\mathbf{x}}) = 0, \tag{2.12}$$

where $\overline{\mathbf{x}} \triangleq \mathbb{1} - \mathbf{x}$.

Proof. We have that

$$\begin{aligned} g_{\mathbf{v}}(r, \mathbf{x}) =& v_r x_1 + \sum_{t=2}^{s-1} (v_{t+r-1} - v_{t+r-2}) x_t - v_{r+s-2} x_s \\ =& v_r x_1 + \sum_{t=2}^{s-1} (v_{t+r-1} - v_{t+r-2}) x_t - v_{r+s-2} x_s - \\ & v_r - \sum_{t=2}^{s-1} (v_{t+r-1} - v_{t+r-2}) + v_{r+s-2} \\ =& v_r (x_1 - 1) + \sum_{t=2}^{s-1} (v_{t+r-1} - v_{t+r-2})(x_t - 1) - \\ & v_{r+s-2}(x_s - 1) = -g_{\mathbf{v}}(r, \overline{\mathbf{x}}), \end{aligned}$$

which proves Eq. (2.12). □

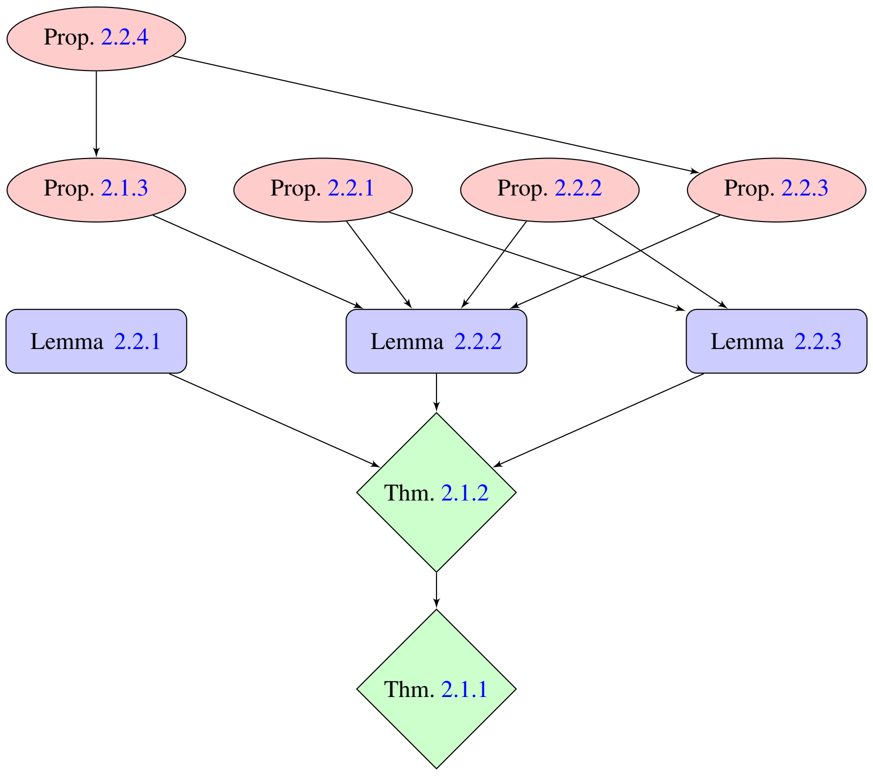

Figure 2.1: Dependencies of the claims in Ch. 2.

Lemma 2.2.2 is more involved compared to Lemma 2.2.3 and is proved in Sec. 2.3, with the proofs of Proposition 2.1.3 and Proposition 2.2.3 given at the end of Sec. 2.3. Lemma 2.2.3 is proved in Sec. 2.4. Finally, the decoding algorithm of Construction 2.1.1 is presented in Sec. 2.5. For the convenience of the reader, a graph of dependencies is given in Figure 3.2.

2.3 Protecting $\mathbb{1}_{10}$ Indicator Vectors

In this section, we prove Lemma 2.2.2. The proof is based on Proposition 2.1.3 and Proposition 2.2.3, which are proved at the end of this section. Since $\mathbf{c} \in B_2(\mathbf{c}')$ it follows that there exist integers i_1, i_2, j_1, and j_2 such that

$$\mathbf{c} \xrightarrow{\text{del' } i_1} \mathbf{d} \xrightarrow{\text{del' } j_1} \mathbf{e}$$
$$\mathbf{c}' \xrightarrow{\text{del' } i_2} \mathbf{d}' \xrightarrow{\text{del' } j_2} \mathbf{e},$$

and by Proposition 2.2.1 it follows that there exist integers ℓ_1, ℓ_2, k_1, and k_2 such that

$$\mathbb{1}_{10}(\mathbf{c}) \xrightarrow{\text{del' } \ell_1} \mathbb{1}_{10}(\mathbf{d}) \xrightarrow{\text{del' } k_1} \mathbb{1}_{10}(\mathbf{e})$$
$$\mathbb{1}_{10}(\mathbf{c}') \xrightarrow{\text{del' } \ell_2} \mathbb{1}_{10}(\mathbf{d}') \xrightarrow{\text{del' } k_2} \mathbb{1}_{10}(\mathbf{e}).$$

Due to symmetry between $\mathbf{c}$ and $\mathbf{c}'$, we distinguish between the following three cases. In each case, the difference between the f values of $\mathbf{c}$ and $\mathbf{c}'$ are given in terms of the function g (2.4). Further, the computation of these three differences, a somewhat tedious but straightforward task, is deferred to the appendix of this chapter.

Case (a). If $\ell_1 \leq \ell_2 < k_2 \leq k_1$ (Fig. 2.2), then

$$\begin{aligned}
\mathbb{1}_{10}(\mathbf{c})_t &= \mathbb{1}_{10}(\mathbf{c}')_t && \text{if } t < \ell_1 \\
& && \text{or } \ell_2 < t < k_2 \\
& && \text{or } t > k_1, \\
\mathbb{1}_{10}(\mathbf{c})_{t+1} &= \mathbb{1}_{10}(\mathbf{c}')_t && \text{if } \ell_1 \leq t \leq \ell_2 - 1, \\
\mathbb{1}_{10}(\mathbf{c})_t &= \mathbb{1}_{10}(\mathbf{c}')_{t+1} && \text{if } k_2 \leq t \leq k_1 - 1.
\end{aligned}$$

Thus, for $e \in \{0, 1, 2\}$,

$$\begin{aligned}
&(\mathbb{1}_{10}(\mathbf{c}) - \mathbb{1}_{10}(\mathbf{c}')) \cdot \mathbf{m}^{(e)} \\
&= g_{\mathbf{m}^{(e)}}(\ell_1, (\mathbb{1}_{10}(\mathbf{c})^{(\ell_1,\ell_2)}, \mathbb{1}_{10}(\mathbf{c}')_{\ell_2})) - \\
&\quad g_{\mathbf{m}^{(e)}}(k_2, (\mathbb{1}_{10}(\mathbf{c}')^{(k_2,k_1)}, \mathbb{1}_{10}(\mathbf{c})_{k_1})).
\end{aligned} \tag{2.13}$$

Case (b). If $\ell_1 \leq \ell_2 < k_1 \leq k_2$ (Fig. 2.3), then

$$\begin{aligned}
\mathbb{1}_{10}(\mathbf{c})_t &= \mathbb{1}_{10}(\mathbf{c}')_t && \text{if } t < l_1 \\
& && \text{or } l_2 < t < k_1. \\
& && \text{or } t > k_2. \\
\mathbb{1}_{10}(\mathbf{c})_{t+1} &= \mathbb{1}_{10}(\mathbf{c}')_t && \text{if } \ell_1 \leq t \leq \ell_2 - 1 \\
& && \text{or } k_1 \leq t \leq k_2 - 1.
\end{aligned}$$

Thus, for $e \in \{0, 1, 2\}$,

$$\begin{aligned}
&(\mathbb{1}_{10}(\mathbf{c}) - \mathbb{1}_{10}(\mathbf{c}')) \cdot \mathbf{m}^{(e)} \\
&= g_{\mathbf{m}^{(e)}}(\ell_1, (\mathbb{1}_{10}(\mathbf{c})^{(\ell_1,\ell_2)}, \mathbb{1}_{10}(\mathbf{c}')_{\ell_2})) + \\
&\quad g_{\mathbf{m}^{(e)}}(k_1, (\mathbb{1}_{10}(\mathbf{c})^{(k_1,k_2)}, \mathbb{1}_{10}(\mathbf{c}')_{k_2})).
\end{aligned} \tag{2.14}$$

Case (c). If $\ell_1 < k_1 \leq \ell_2 < k_2$ (Fig. 2.4), then

$$\begin{aligned}
\mathbb{1}_{10}(\mathbf{c})_t &= \mathbb{1}_{10}(\mathbf{c}')_t && \text{if } t < l_1 \\
& && \text{or } t > k_2,
\end{aligned}$$

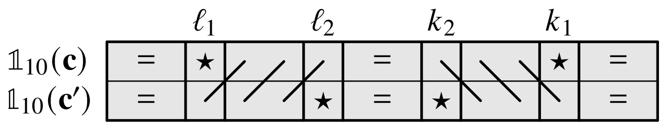

Figure 2.2: Case (a), where $\ell_1 \leq \ell_2 < k_2 \leq k_1$. The diagonal lines indicate equality between the respective entries of $\mathbb{1}_{10}(\mathbf{c})$ and $\mathbb{1}_{10}(\mathbf{c}')$.

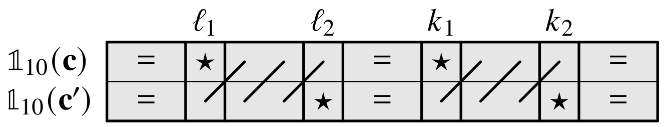

Figure 2.3: Case (b), where $\ell_1 \leq \ell_2 < k_1 \leq k_2$. The diagonal lines indicate equality between the respective entries of $\mathbb{1}_{10}(\mathbf{c})$ and $\mathbb{1}_{10}(\mathbf{c}')$.

$$\begin{aligned} \mathbb{1}_{10}(\mathbf{c})_{t+1} &= \mathbb{1}_{10}(\mathbf{c}')_t && \text{if } \ell_1 \leq t \leq k_1 - 2 \\ & && \text{or } \ell_2 + 1 \leq t \leq k_2 - 1, \\ \mathbb{1}_{10}(\mathbf{c})_{t+2} &= \mathbb{1}_{10}(\mathbf{c}')_t && \text{if } k_1 - 1 \leq t \leq \ell_2 - 1. \end{aligned}$$

Thus, for $e \in \{0, 1, 2\}$,

$$\begin{aligned} &(\mathbb{1}_{10}(\mathbf{c}) - \mathbb{1}_{10}(\mathbf{c}')) \cdot \mathbf{m}^{(e)} \\ =\, & g_{\mathbf{m}^{(e)}}(\ell_1, (\mathbb{1}_{10}(\mathbf{c})^{(\ell_1, k_1-1)}, \mathbb{1}_{10}(\mathbf{c})^{(k_1+1, \ell_2+1)}, \mathbb{1}_{10}(\mathbf{c}')_{\ell_2}))+ \\ & g_{\mathbf{m}^{(e)}}(k_1, (\mathbb{1}_{10}(\mathbf{c})^{(k_1, k_2)}, \mathbb{1}_{10}(\mathbf{c}')_{k_2})). \end{aligned} \tag{2.15}$$

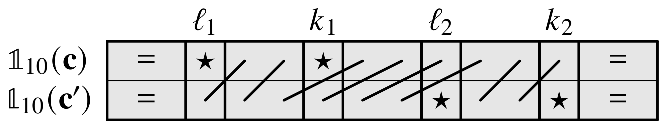

Figure 2.4: Case (c), where $\ell_1 < k_1 \leq \ell_2 < k_2$. The diagonal lines indicate equality between the respective entries of $\mathbb{1}_{10}(\mathbf{c})$ and $\mathbb{1}_{10}(\mathbf{c}')$.

Note that if $f(\mathbf{c}) = f(\mathbf{c}')$, then $\mathbb{1}_{10}(\mathbf{c}) \cdot \mathbf{m}^{(e)} \equiv \mathbb{1}_{10}(\mathbf{c}') \cdot \mathbf{m}^{(e)} \bmod n_e$, where $n_0 = 2n, n_1 = n^2$, and $n_2 = n^3$. Hence, from (2.13)–(2.15) we have that

$$\begin{aligned} & g_{\mathbf{m}^{(e)}}(\ell_1, (\mathbb{1}_{10}(\mathbf{c})^{(\ell_1, \ell_2)}, \mathbb{1}_{10}(\mathbf{c}')_{\ell_2})) \\ -& g_{\mathbf{m}^{(e)}}(k_2, (\mathbb{1}_{10}(\mathbf{c}')^{(k_2, k_1)}, \mathbb{1}_{10}(\mathbf{c})_{k_1})) \equiv 0 \bmod n_e, \end{aligned}$$

$$\begin{aligned}
&g_{\mathbf{m}^{(e)}}(\ell_1, (\mathbb{1}_{10}(\mathbf{c})^{(\ell_1,\ell_2)}, \mathbb{1}_{10}(\mathbf{c}')_{\ell_2})) \\
&+g_{\mathbf{m}^{(e)}}(k_1, (\mathbb{1}_{10}(\mathbf{c})^{(k_1,k_2)}, \mathbb{1}_{10}(\mathbf{c}')_{k_2})) \equiv 0 \bmod n_e, \text{ and} \\
&g_{\mathbf{m}^{(e)}}(\ell_1, (\mathbb{1}_{10}(\mathbf{c})^{(\ell_1,k_1-1)}, \mathbb{1}_{10}(\mathbf{c})^{(k_1+1,\ell_2+1)}, \mathbb{1}_{10}(\mathbf{c}')_{\ell_2})) \\
&+g_{\mathbf{m}^{(e)}}(k_1, (\mathbb{1}_{10}(\mathbf{c})^{(k_1,k_2)}, \mathbb{1}_{10}(\mathbf{c}')_{k_2})) \equiv 0 \bmod n_e, \qquad (2.16)
\end{aligned}$$

for Case (a), Case (b), and Case (c), respectively.

In what follows, we show that these equalities also hold in their non modular version. Specifically, we prove the following,

$$\begin{aligned}
&g_{\mathbf{m}^{(e)}}(\ell_1, (\mathbb{1}_{10}(\mathbf{c})^{(\ell_1,\ell_2)}, \mathbb{1}_{10}(\mathbf{c}')_{\ell_2})) \\
&-g_{\mathbf{m}^{(e)}}(k_2, (\mathbb{1}_{10}(\mathbf{c}')^{(k_2,k_1)}, \mathbb{1}_{10}(\mathbf{c})_{k_1})) = 0, \qquad (2.17) \\
&g_{\mathbf{m}^{(e)}}(\ell_1, (\mathbb{1}_{10}(\mathbf{c})^{(\ell_1,\ell_2)}, \mathbb{1}_{10}(\mathbf{c}')_{\ell_2})) \\
&+g_{\mathbf{m}^{(e)}}(k_1, (\mathbb{1}_{10}(\mathbf{c})^{(k_1,k_2)}, \mathbb{1}_{10}(\mathbf{c}')_{k_2})) = 0, \text{ and} \qquad (2.18) \\
&g_{\mathbf{m}^{(e)}}(\ell_1, (\mathbb{1}_{10}(\mathbf{c})^{(\ell_1,k_1-1)}, \mathbb{1}_{10}(\mathbf{c})^{(k_1+1,\ell_2+1)}, \mathbb{1}_{10}(\mathbf{c}')_{\ell_2})) \\
&+g_{\mathbf{m}^{(e)}}(k_1, (\mathbb{1}_{10}(\mathbf{c})^{(k_1,k_2)}, \mathbb{1}_{10}(\mathbf{c}')_{k_2})) = 0. \qquad (2.19)
\end{aligned}$$

From (2.8), we have that

$$-\mathbf{m}^{(e)}_{r+s-2} \le g_{\mathbf{m}^{(e)}}(r, \mathbf{x}) \le \mathbf{m}^{(e)}_{r+s-2}$$

for any $\mathbf{x} \in \{0,1\}^s$ and any integer r that satisfies $r + s - 2 \le n - 1$. Therefore,

$$\begin{aligned}
-\mathbf{m}^{(e)}_{\ell_2} - \mathbf{m}^{(e)}_{k_1} \le\, & g_{\mathbf{m}^{(e)}}(\ell_1, (\mathbb{1}_{10}(\mathbf{c})^{(\ell_1,\ell_2)}, \mathbb{1}_{10}(\mathbf{c}')_{\ell_2})) - \\
& g_{\mathbf{m}^{(e)}}(k_2, (\mathbb{1}_{10}(\mathbf{c}')^{(k_2,k_1)}, \mathbb{1}_{10}(\mathbf{c})_{k_1})) \\
\le\, & \mathbf{m}^{(e)}_{\ell_2} + \mathbf{m}^{(e)}_{k_1}, \\
-\mathbf{m}^{(e)}_{\ell_2} - \mathbf{m}^{(e)}_{k_2} \le\, & g_{\mathbf{m}^{(e)}}(\ell_1, (\mathbb{1}_{10}(\mathbf{c})^{(\ell_1,\ell_2)}, \mathbb{1}_{10}(\mathbf{c}')_{\ell_2})) + \\
& g_{\mathbf{m}^{(e)}}(k_1, (\mathbb{1}_{10}(\mathbf{c})^{(k_1,k_2)}, \mathbb{1}_{10}(\mathbf{c}')_{k_2})) \\
\le\, & \mathbf{m}^{(e)}_{\ell_2} + \mathbf{m}^{(e)}_{k_2}, \text{ and} \\
-\mathbf{m}^{(e)}_{\ell_2} - \mathbf{m}^{(e)}_{k_2} \le\, & g_{\mathbf{m}^{(e)}}(\ell_1, (\mathbb{1}_{10}(\mathbf{c})^{(\ell_1,k_1-1)}, \\
& \mathbb{1}_{10}(\mathbf{c})^{(k_1+1,\ell_2+1)}, \mathbb{1}_{10}(\mathbf{c}')_{\ell_2})) \\
& + g_{\mathbf{m}^{(e)}}(k_1, (\mathbb{1}_{10}(\mathbf{c})^{(k_1,k_2)}, \mathbb{1}_{10}(\mathbf{c}')_{k_2})) \\
\le\, & \mathbf{m}^{(e)}_{\ell_2} + \mathbf{m}^{(e)}_{k_2}, \qquad (2.20)
\end{aligned}$$

for Case (a)–(c) respectively. Further, note that

$$\mathbf{m}^{(e)}_{\ell_2} + \mathbf{m}^{(e)}_{k_1} < n_e \text{ and } \mathbf{m}^{(e)}_{\ell_2} + \mathbf{m}^{(e)}_{k_2} < n_e. \qquad (2.21)$$

Combining (2.16), (2.20), and (2.21), we conclude that if $f(\mathbf{c}) = f(\mathbf{c}')$, then Eq. (2.17), (2.18), and (2.19) hold for Case (a), Case (b), and Case (c) respectively.

For Case (a), Equation (2.17) and Proposition 2.1.3 imply that

$$\mathbb{1}_{10}(\mathbf{c})_{\ell_1} = \ldots = \mathbb{1}_{10}(\mathbf{c})_{\ell_2} = \mathbb{1}_{10}(\mathbf{c}')_{\ell_2}$$
$$\mathbb{1}_{10}(\mathbf{c}')_{k_2} = \ldots = \mathbb{1}_{10}(\mathbf{c}')_{k_1} = \mathbb{1}_{10}(\mathbf{c})_{k_1},$$

which readily implies that

$$\mathbb{1}_{10}(\mathbf{c}')_t = \mathbb{1}_{10}(\mathbf{c})_{t+1} = \mathbb{1}_{10}(\mathbf{c})_t$$

for $\ell_1 \leq t < \ell_2$ and

$$\mathbb{1}_{10}(\mathbf{c})_t = \mathbb{1}_{10}(\mathbf{c}')_{t+1} = \mathbb{1}_{10}(\mathbf{c}')_t$$

for $k_2 \leq t < k_1$. Together with $\mathbb{1}_{10}(\mathbf{c})_{\ell_2} = \mathbb{1}_{10}(\mathbf{c}')_{\ell_2}$ and $\mathbb{1}_{10}(\mathbf{c}')_{k_1} = \mathbb{1}_{10}(\mathbf{c})_{k_1}$, we have that $\mathbb{1}_{10}(\mathbf{c}) = \mathbb{1}_{10}(\mathbf{c}')$.

For Case (b), Equation (2.18) and Proposition 2.1.3 implies that

$$\mathbb{1}_{10}(\mathbf{c})_{\ell_1} = \ldots = \mathbb{1}_{10}(\mathbf{c})_{\ell_2} = \mathbb{1}_{10}(\mathbf{c}')_{\ell_2}$$
$$\mathbb{1}_{10}(\mathbf{c}')_{k_1} = \ldots = \mathbb{1}_{10}(\mathbf{c}')_{k_2} = \mathbb{1}_{10}(\mathbf{c})_{k_2}$$

and hence

$$\mathbb{1}_{10}(\mathbf{c}')_t = \mathbb{1}_{10}(\mathbf{c})_{t+1} = \mathbb{1}_{10}(\mathbf{c})_t$$

for $\ell_1 \leq t < \ell_2$ and $k_1 \leq t < k_2$, and thus $\mathbb{1}_{10}(\mathbf{c}) = \mathbb{1}_{10}(\mathbf{c}')$.

Note that in Case (a) and (b) we used Proposition 2.1.3, which implies that only two parity checks with weights $\mathbf{m}^{(0)}$ and $\mathbf{m}^{(1)}$ are needed. Case (c) is the most involved case and the only case when the parity check with weight $\mathbf{m}^{(2)}$ is needed. Hence, Proposition 2.2.3 is required. According to Equation (2.19) and Proposition 2.2.3, we have either

$$\mathbb{1}_{10}(\mathbf{c})_{\ell_1} = \ldots = \mathbb{1}_{10}(\mathbf{c})_{k_2} = \mathbb{1}_{10}(\mathbf{c}')_{\ell_2} = \mathbb{1}_{10}(\mathbf{c}')_{k_2} \tag{2.22}$$

or

$$\mathbb{1}_{10}(\mathbf{c})_{\ell_1} = \ldots = \mathbb{1}_{10}(\mathbf{c})_{k_1-1} = \mathbb{1}_{10}(\mathbf{c})_{k_1+1},$$
$$\mathbb{1}_{10}(\mathbf{c})_i + \mathbb{1}_{10}(\mathbf{c})_{i+1} = 1 \text{ for } i \in \{k_1, \ldots, \ell_2\},$$

$$\mathbb{1}_{10}(\mathbf{c}')_{\ell_2} + \mathbb{1}_{10}(\mathbf{c}')_{k_2} = 1, \text{ and}$$
$$\mathbb{1}_{10}(\mathbf{c})_{\ell_2+1} = \ldots = \mathbb{1}_{10}(\mathbf{c})_{k_2} = \mathbb{1}_{10}(\mathbf{c}')_{k_2}. \tag{2.23}$$

If (2.22) is true, we can obtain $\mathbf{c} = \mathbf{c}'$ by following steps similar to those of Case (a) and Case (b). If (2.23) is true, we have

$$\mathbb{1}_{10}(\mathbf{c}')_t = \mathbb{1}_{10}(\mathbf{c})_{t+1} = \mathbb{1}_{10}(\mathbf{c})_t$$

for $\ell_1 \leq t \leq k_1 - 2$ and $\ell_2 + 1 \leq t \leq k_2 - 1$. Furthermore, we have that

$$\mathbb{1}_{10}(\mathbf{c}')_t = \mathbb{1}_{10}(\mathbf{c})_{t+2} = 1 - \mathbb{1}_{10}(\mathbf{c})_{t+1} = \mathbb{1}_{10}(\mathbf{c})_t$$

for $k_1 \leq t \leq \ell_2 - 1$. In addition, we have $\mathbb{1}_{10}(\mathbf{c}')_{k_1-1} = \mathbb{1}_{10}(\mathbf{c})_{k_1+1} = \mathbb{1}_{10}(\mathbf{c})_{k_1-1}, \mathbb{1}_{10}(\mathbf{c}')_{\ell_2} = 1 - \mathbb{1}_{10}(\mathbf{c}')_{k_2} = 1 - \mathbb{1}_{10}(\mathbf{c})_{\ell_2+1} = \mathbb{1}_{10}(\mathbf{c})_{\ell_2}$ and $\mathbb{1}_{10}(\mathbf{c}')_{k_2} = \mathbb{1}_{10}(\mathbf{c})_{k_2}$. Hence, we have that $\mathbb{1}_{10}(\mathbf{c}) = \mathbb{1}_{10}(\mathbf{c}')$. This concludes the proof of Cases (a), (b), and (c), and thus the proof of Lemma 2.2.2 is completed.

We now prove Proposition 2.1.3 and Proposition 2.2.3.

Proof. (of Proposition 2.1.3) According to Eq. (2.12), if $\lambda = 1$, then Eq. (2.9) can be written as

$$g_{\mathbf{m}^{(0)}}(r_1, \mathbf{x}) - g_{\mathbf{m}^{(0)}}(r_2, \overline{\mathbf{y}}) = 0, \text{ and}$$
$$g_{\mathbf{m}^{(1)}}(r_1, \mathbf{x}) - g_{\mathbf{m}^{(1)}}(r_2, \overline{\mathbf{y}}) = 0.$$

Therefore, it suffices to prove the claim for $\lambda = -1$. We distinguish between four cases according to the value of (y_1, y_{s_2}).

Case (a). $(y_1, y_{s_2}) = (0, 1)$. We have that

$$\begin{aligned}
& g_{\mathbf{m}^{(e)}}(r_1, \mathbf{x}) - g_{\mathbf{m}^{(e)}}(r_2, \mathbf{y}) \\
= {} & \mathbf{m}^{(e)}_{r_1} x_1 + \sum_{t=2}^{s_1-1} (\mathbf{m}^{(e)}_{t+r_1-1} - \mathbf{m}^{(e)}_{t+r_1-2}) x_t - \mathbf{m}^{(e)}_{r_1+s_1-2} x_{s_1} - \\
& \mathbf{m}^{(e)}_{r_2} y_1 - \sum_{t=2}^{s_2-1} (\mathbf{m}^{(e)}_{t+r_2-1} - \mathbf{m}^{(e)}_{t+r_2-2}) y_t + \mathbf{m}^{(e)}_{r_2+s_2-2} y_{s_2} \\
\geq {} & -\mathbf{m}^{(e)}_{r_1+s_1-2} - \sum_{t=2}^{s_2-1} (\mathbf{m}^{(e)}_{t+r_2-1} - \mathbf{m}^{(e)}_{t+r_2-2}) + \mathbf{m}^{(e)}_{r_2+s_2-2} \\
= {} & \mathbf{m}^{(e)}_{r_2} - \mathbf{m}^{(e)}_{r_1+s_1-2} > 0,
\end{aligned}$$

a contradiction.

Case (b). $(y_1, y_{s_2}) = (1, 0)$. From Proposition 2.2.4 and (2.9) we have $g_{\mathbf{m}^{(e)}}(r_1, \overline{\mathbf{x}}) - g_{\mathbf{m}^{(e)}}(r_2, \overline{\mathbf{y}}) = 0$ for $e \in \{0, 1\}$, where $\overline{\mathbf{x}} \triangleq \mathbb{1} - \mathbf{x}$ and $\overline{\mathbf{y}} \triangleq \mathbb{1} - \mathbf{y}$. Since $(\overline{y}_1, \overline{y}_{s_2}) = (0, 1)$, from the previous case this leads to a contradiction.

Case (c). $(y_1, y_{s_2}) = (1, 1)$. Let

$$\begin{aligned} S_1 &\triangleq \{j : y_{j-r_2+1} = 1, r_2 + 1 \le j \le r_2 + s_2 - 2\}, \text{ and} \\ S_1^c &\triangleq \{j : y_{j-r_2+1} = 0, r_2 + 1 \le j \le r_2 + s_2 - 2\}, \end{aligned}$$

and notice that

$$\begin{aligned} & g_{\mathbf{m}^{(0)}}(r_2, \mathbf{y}) \\ &= \mathbf{m}^{(0)}_{r_2} - \mathbf{m}^{(0)}_{r_2+s_2-2} + \sum_{j=2}^{s_2-1} (\mathbf{m}^{(0)}_{j+r_2-1} - \mathbf{m}^{(0)}_{j+r_2-2}) y_j \\ &= -\sum_{j=r_2+1}^{r_2+s_2-2} (\mathbf{m}^{(0)}_{j} - \mathbf{m}^{(0)}_{j-1}) + \sum_{j=2}^{s_2-1} (\mathbf{m}^{(0)}_{j+r_2-1} - \mathbf{m}^{(0)}_{j+r_2-2}) y_j \\ &= -\sum_{j=r_2+1}^{r_2+s_2-2} (\mathbf{m}^{(0)}_{j} - \mathbf{m}^{(0)}_{j-1})(1 - y_j) \\ &= -\sum_{j \in S_1^c} (\mathbf{m}^{(0)}_{j} - \mathbf{m}^{(0)}_{j-1}) = -\sum_{j \in S_1^c} 1, \text{ and similarly,} \\ & g_{\mathbf{m}^{(1)}}(r_2, \mathbf{y}) \\ &= -\sum_{j \in S_1^c} (\mathbf{m}^{(1)}_{j} - \mathbf{m}^{(1)}_{j-1}) = -\sum_{j \in S_1^c} j. \end{aligned} \tag{2.24}$$

Now, on the one hand, if $x_{s_1} = 0$ we have

$$\begin{aligned} g_{\mathbf{m}^{(0)}}(r_1, \mathbf{x}) =& \mathbf{m}^{(0)}_{r_1} x_1 + \sum_{t=2}^{s_1-1} (\mathbf{m}^{(0)}_{t+r_1-1} - \mathbf{m}^{(0)}_{t+r_1-2}) x_t \\ \ge& 0, \end{aligned} \tag{2.25}$$

and hence, (2.24) and (2.25) imply that $g_{\mathbf{m}^{(0)}}(r_1, \mathbf{x}) - g_{\mathbf{m}^{(0)}}(r_2, \mathbf{y}) \ge 0$, and equality holds only when $g_{\mathbf{m}^{(0)}}(r_1, \mathbf{x})$ and $g_{\mathbf{m}^{(0)}}(r_2, \mathbf{y})$ are both 0, which by Proposition 2.1.2 implies that $\mathbf{x}$ and $\mathbf{y}$ are constant vectors. On the other hand, if $x_{s_1} = 1$ let $S_2 = \{t : x_{\max\{t-r_1+1,1\}} = 0, 1 \le t \le r_1 + s_1 - 2\}$, and notice that

$$\begin{aligned} & g_{\mathbf{m}^{(0)}}(r_1, \mathbf{x}) \\ &= \mathbf{m}^{(0)}_{r_1} x_1 + \sum_{t=2}^{s_1-1} (\mathbf{m}^{(0)}_{t+r_1-1} - \mathbf{m}^{(0)}_{t+r_1-2}) x_t - \mathbf{m}^{(0)}_{r_1+s_1-2} \end{aligned}$$

$$= \mathbf{m}^{(0)}_{r_1}(x_1 - 1) + \sum_{t=2}^{s_1-1}(\mathbf{m}^{(0)}_{t+r_1-1} - \mathbf{m}^{(0)}_{t+r_1-2})(x_t - 1)$$
$$= -\sum_{t\in S_2} 1, \text{ and similarly,}$$
$$g_{\mathbf{m}^{(1)}}(r_1, \mathbf{x}) = -\sum_{t\in S_2} t. \tag{2.26}$$

Inserting (2.24) and (2.26) into (2.9), we have

$$-\sum_{t\in S_2} 1 + \sum_{j\in S_1^c} 1 = 0,$$
$$-\sum_{t\in S_2} t + \sum_{j\in S_1^c} j = 0.$$

This implies that the sets S_1^c and S_2 have the same cardinality and the same sum of elements. However, the maximum element in S_2 is smaller than the minimum element in S_1^c. Therefore, S_1^c and S_2 are empty, which implies that $\mathbf{x}$ is the 0 vector and $\mathbf{y}$ is the all 1's vector.

Case (d). $(y_1, y_{s_2}) = (0, 0)$. From Proposition 2.2.4 and Eq. (2.9) we have $g_{\mathbf{m}^{(e)}}(r_1, \overline{\mathbf{x}}) - g_{\mathbf{m}^{(e)}}(r_2, \overline{\mathbf{y}}) = 0$ for $e \in \{0, 1\}$, where $\overline{\mathbf{x}} \triangleq \mathbb{1} - \mathbf{x}$ and $\overline{\mathbf{y}} \triangleq \mathbb{1} - \mathbf{y}$. Since $(\overline{y}_1, \overline{y}_{s_2}) = (1, 1)$, from the previous case $\overline{\mathbf{x}}$ and $\overline{\mathbf{y}}$ are constant vectors, and thus so are $\mathbf{x}$ and $\mathbf{y}$. □

Proof. (of Proposition 2.2.3) We distinguish between four cases according to the value of $(x_{s_1+s_2+1}, y_{s_2+s_3+1})$.

Case (a). $(x_{s_1+s_2+1}, y_{s_2+s_3+1}) = (0, 0)$. Similar to (2.25), we have that $g_{\mathbf{m}^{(0)}}(r_1, \mathbf{x}) + g_{\mathbf{m}^{(0)}}(r_2, \mathbf{y}) \geq 0$, where equality holds only if $\mathbf{x}$ and $\mathbf{y}$ are constant 0 vectors.

Case (b). $(x_{s_1+s_2+1}, y_{s_2+s_3+1}) = (1, 1)$. From Proposition 2.2.4 and Eq. (2.10) we have that $g_{\mathbf{m}^{(0)}}(r_1, \overline{\mathbf{x}}) + g_{\mathbf{m}^{(0)}}(r_2, \overline{\mathbf{y}}) = 0$. On the other hand, since $(\overline{x}_{s_1+s_2+1}, \overline{y}_{s_2+s_3+1}) = (0, 0)$, it follows that $g_{\mathbf{m}^{(0)}}(r_1, \overline{\mathbf{x}}) + g_{\mathbf{m}^{(0)}}(r_2, \overline{\mathbf{y}}) \geq 0$ where equality holds when $\mathbf{x}$ and $\mathbf{y}$ are constant 1 vectors.

Case (c). $(x_{s_1+s_2+1}, y_{s_2+s_3+1}) = (0, 1)$. On the one hand, for $y_1 = 0$ we have

$$g_{\mathbf{m}^{(0)}}(r_1, \mathbf{x}) + g_{\mathbf{m}^{(0)}}(r_2, \mathbf{y})$$
$$= \mathbf{m}^{(0)}_{r_1} x_1 + \sum_{t=2}^{s_1+1}(\mathbf{m}^{(0)}_{t+r_1-1} - \mathbf{m}^{(0)}_{t+r_1-2})x_t +$$

$$\begin{aligned}
&\sum_{t=s_1+1}^{s_1+s_2-1} (\mathbf{m}^{(0)}_{t+r_1} - \mathbf{m}^{(0)}_{t+r_1-1})x_{t+1} \\
&+ \sum_{t=2}^{s_2} (\mathbf{m}^{(0)}_{t+r_2-1} - \mathbf{m}^{(0)}_{t+r_2-2})y_t + \\
&\sum_{t=s_2+1}^{s_2+s_3} (\mathbf{m}^{(0)}_{t+r_2-1} - \mathbf{m}^{(0)}_{t+r_2-2})y_t - \mathbf{m}^{(0)}_{r_2+s_2+s_3-1} \\
= {}& \mathbf{m}^{(0)}_{r_1} x_1 + \sum_{t=2}^{s_1+1} (\mathbf{m}^{(0)}_{t+r_1-1} - \mathbf{m}^{(0)}_{t+r_1-2})x_t + \\
&\sum_{t=s_1+1}^{s_1+s_2-1} (\mathbf{m}^{(0)}_{t+r_1} - \mathbf{m}^{(0)}_{t+r_1-1})(x_t + x_{t+1}) \\
&+ \sum_{t=s_2+1}^{s_2+s_3} (\mathbf{m}^{(0)}_{t+r_2-1} - \mathbf{m}^{(0)}_{t+r_2-2})y_t - \mathbf{m}^{(0)}_{r_2+s_2+s_3-1} \\
\le {}& \mathbf{m}^{(0)}_{r_1} + \sum_{t=2}^{s_1+1} (\mathbf{m}^{(0)}_{t+r_1-1} - \mathbf{m}^{(0)}_{t+r_1-2}) + \\
&\sum_{t=s_1+1}^{s_1+s_2-1} (\mathbf{m}^{(0)}_{t+r_1} - \mathbf{m}^{(0)}_{t+r_1-1}) \\
&+ \sum_{t=s_2+1}^{s_2+s_3} (\mathbf{m}^{(0)}_{t+r_2-1} - \mathbf{m}^{(0)}_{t+r_2-2}) - \mathbf{m}^{(0)}_{r_2+s_2+s_3-1} = 0,
\end{aligned}$$

where equality equality holds when

$$\begin{aligned}
&x_t = 1 \text{ for } t \in \{1, \dots, s_1 + 1\}, \\
&x_t + x_{t+1} = 1 \text{ for } t \in \{s_1 + 1, \dots, s_1 + s_2 - 1\}, \text{ and} \\
&y_t = 1 \text{ for } t \in \{s_2 + 1, \dots, s_2 + s_3\},
\end{aligned}$$

and hence (2.11) holds. On the other hand, when $y_1 = 1$, let

$$\begin{aligned}
S_1 &= \{t : x_{\max\{t-r_1+1,1\}} = 1, 1 \le t \le s_1 + r_1\}, \\
S_2 &= \{t : x_{t-r_1} + x_{t-r_1+1} = 0, r_2 + 1 \le t \le r_2 + s_2 - 1\}, \\
S_3 &= \{t : y_{t-r_2+1} = 0, r_2 + s_2 \le t \le r_2 + s_2 + s_3 - 1\},
\end{aligned}$$

and notice that

$$\begin{aligned}
&g_{\mathbf{m}^{(0)}}(r_1, \mathbf{x}) + g_{\mathbf{m}^{(0)}}(r_2, \mathbf{y}) \\
= {}& \mathbf{m}^{(0)}_{r_1} x_1 + \sum_{t=2}^{s_1+1} (\mathbf{m}^{(0)}_{t+r_1-1} - \mathbf{m}^{(0)}_{t+r_1-2})x_t +
\end{aligned}$$

$$\begin{aligned}
&\mathbf{m}^{(0)}_{s_1+r_1} + \sum_{t=s_1+1}^{s_1+s_2-1} (\mathbf{m}^{(0)}_{t+r_1} - \mathbf{m}^{(0)}_{t+r_1-1})(x_t + x_{t+1}) + \\
&\sum_{t=s_2+1}^{s_2+s_3} (\mathbf{m}^{(0)}_{t+r_2-1} - \mathbf{m}^{(0)}_{t+r_1-2}) y_t - \mathbf{m}^{(0)}_{r_2+s_2+s_3-1} \\
&= \sum_{t\in S_1} (\mathbf{m}^{(0)}_t - \mathbf{m}^{(0)}_{t-1}) - \sum_{t\in S_2} (\mathbf{m}^{(0)}_t - \mathbf{m}^{(0)}_{t-1}) - \\
&\sum_{t\in S_3} (\mathbf{m}^{(0)}_t - \mathbf{m}^{(0)}_{t-1}) \\
&= \sum_{t\in S_1} 1 - \sum_{t\in S_2} 1 - \sum_{t\in S_3} 1. \qquad (2.27)
\end{aligned}$$

Similarly, we have that

$$g_{\mathbf{m}^{(1)}}(r_1, \mathbf{x}) + g_{\mathbf{m}^{(1)}}(r_2, \mathbf{y}) = \sum_{t\in S_1} t - \sum_{t\in S_2} t - \sum_{t\in S_3} t. \qquad (2.28)$$

Equations (2.10), (2.27), and (2.28) imply that the cardinality of S_1 equals the sum of cardinalities of S_2 and S_3, and in addition, the sum of elements of S_1 equals the sum of elements of S_2 and S_3. Note that the minimum element of $S_2 \cup S_3$ is larger than the maximum element of S_1. This is impossible, unless S_1, S_2, and S_3 are empty, which implies that $x_t = 0$ for $t \in \{1, \ldots, s_1+1\}$, $x_t+x_{t+1} = 1$ for $t \in \{s_1+1, \ldots, s_1+s_2-1\}$, and $y_t = 1$ for $t \in \{s_2 + 1, \ldots, s_2 + s_3\}$, and hence (2.11) holds.

Case (d). $(x_{s_1+s_2+1}, y_{s_2+s_3+1}) = (1, 0)$. On the one hand, for $y_1 = 0$, let

$$\begin{aligned}
S_1 &= \{t : x_{\max\{t-r_1+1,1\}} = 0, 1 \le t \le s_1 + r_1\}, \\
S_2 &= \{t : x_{t-r_1} + x_{t-r_1+1} = 0, r_2 + 1 \le t \le r_2 + s_2 - 1\}, \\
S_3 &= \{t : y_{t-r_2+1} = 1, r_2 + s_2 \le t \le r_2 + s_2 + s_3 - 1\}.
\end{aligned}$$

We have

$$\begin{aligned}
&g_{\mathbf{m}^{(0)}}(r_1, \mathbf{x}) + g_{\mathbf{m}^{(0)}}(r_2, \mathbf{y}) \\
&= \mathbf{m}^{(0)}_{r_1} x_1 + \sum_{t=2}^{s_1+1} (\mathbf{m}^{(0)}_{t+r_1-1} - \mathbf{m}^{(0)}_{t+r_1-2}) x_t + \\
&\sum_{t=s_1+1}^{s_1+s_2-1} (\mathbf{m}^{(0)}_{t+r_1} - \mathbf{m}^{(0)}_{t+r_1-1})(x_t + x_{t+1}) - \\
&\mathbf{m}^{(0)}_{r_1+s_1+s_2-1} + \sum_{t=s_2+1}^{s_2+s_3} (\mathbf{m}^{(0)}_{t+r_2-1} - \mathbf{m}^{(0)}_{t+r_1-2}) y_t
\end{aligned}$$

$$
\begin{aligned}
&= -\mathbf{m}^{(0)}_{r_1}(1-x_1) - \sum_{t=2}^{s_1+1}(\mathbf{m}^{(0)}_{t+r_1-1} - \mathbf{m}^{(0)}_{t+r_1-2})(1-x_t) - \\
&\quad \sum_{t=s_1+1}^{s_1+s_2-1}(\mathbf{m}^{(0)}_{t+r_1} - \mathbf{m}^{(0)}_{t+r_1-1})(1-x_t-x_{t+1}) + \\
&\quad \sum_{t=s_2+1}^{s_2+s_3}(\mathbf{m}^{(0)}_{t+r_2-1} - \mathbf{m}^{(0)}_{t+r_1-2})y_t \\
&= -\sum_{t\in S_1}(\mathbf{m}^{(0)}_{t} - \mathbf{m}^{(0)}_{t-1}) - \sum_{t\in S_2}(\mathbf{m}^{(0)}_{t} - \mathbf{m}^{(0)}_{t-1}) + \\
&\quad \sum_{t\in S_3}(\mathbf{m}^{(0)}_{t} - \mathbf{m}^{(0)}_{t-1}) \\
&= -\sum_{t\in S_1}1 - \sum_{t\in S_2}1 + \sum_{t\in S_3}1 = 0.
\end{aligned}
$$

Then similar to the previous case, we obtain sets with identical cardinalities and sum of elements, and yet the smallest element in one is greater than the largest element in the others. Therefore, it follows that S_1, S_2, and S_3 are empty. Then we have $x_t = 1$ for $t \in \{1, \ldots, s_1+1\}$, $x_t + x_{t+1} = 1$ for $t \in \{s_1+1, \ldots, s_1+s_2-1\}$, and $y_t = 0$ for $t \in \{s_2+1, \ldots, s_2+s_3\}$, and hence (2.11) holds.

On the other hand, for $y_1 = 1$, let

$$
\begin{aligned}
S_1 &= \{t : x_{\max\{t-r_1+1,1\}} = 1, 1 \le t \le s_1 + r_1\}, \\
S_2 &= \{t : x_{t-r_1} + x_{t-r_1+1} = 0, r_2+1 \le t \le r_2+s_2-1\}, \\
S_3 &= \{t : y_{t-s_2+1} = 1, r_2+s_2 \le t \le r_2+s_2+s_3-1\}.
\end{aligned}
$$

We have

$$
\begin{aligned}
&g_{\mathbf{m}^{(0)}}(r_1, \mathbf{x}) + g_{\mathbf{m}^{(0)}}(r_2, \mathbf{y}) \\
&= \mathbf{m}^{(0)}_{r_1}x_1 + \sum_{t=2}^{s_1+1}(\mathbf{m}^{(0)}_{t+r_1-1} - \mathbf{m}^{(0)}_{t+r_1-2})x_t + \mathbf{m}^{(0)}_{s_1+r_1} + \\
&\quad \sum_{t=s_1+1}^{s_1+s_2-1}(\mathbf{m}^{(0)}_{t+r_1} - \mathbf{m}^{(0)}_{t+r_1-1})(x_t + x_{t+1}) - \\
&\quad \mathbf{m}^{(0)}_{r_1+s_1+s_2-1} + \sum_{t=s_2+1}^{s_2+s_3}(\mathbf{m}^{(0)}_{t+r_2-1} - \mathbf{m}^{(0)}_{t+r_1-2})y_t \\
&= \mathbf{m}^{(0)}_{r_1}x_1 + \sum_{t=2}^{s_1+1}(\mathbf{m}^{(0)}_{t+r_1-1} - \mathbf{m}^{(0)}_{t+r_1-2})x_t -
\end{aligned}
$$

$$\begin{aligned}
&\sum_{t=s_1+1}^{s_1+s_2-1}(\mathbf{m}^{(0)}_{t+r_1}-\mathbf{m}^{(0)}_{t+r_1-1})(1-x_t-x_{t+1})+\\
&\sum_{t=s_2+1}^{s_2+s_3}(\mathbf{m}^{(0)}_{t+r_2-1}-\mathbf{m}^{(0)}_{t+r_1-2})y_t\\
=&\sum_{t\in S_1}(\mathbf{m}^{(0)}_{t}-\mathbf{m}^{(0)}_{t-1})-\sum_{t\in S_2}(\mathbf{m}^{(0)}_{t}-\mathbf{m}^{(0)}_{t-1})+\\
&\sum_{t\in S_3}(\mathbf{m}^{(0)}_{t}-\mathbf{m}^{(0)}_{t-1})\\
=&\sum_{t\in S_1}1-\sum_{t\in S_2}1+\sum_{t\in S_3}1=0. \qquad (2.29)
\end{aligned}$$

Similarly, we have

$$\begin{aligned}
g_{\mathbf{m}^{(1)}}(r_1,\mathbf{x})+g_{\mathbf{m}^{(1)}}(r_2,\mathbf{y}) &= \sum_{t\in S_1}t-\sum_{t\in S_2}t+\sum_{t\in S_3}t\\
g_{\mathbf{m}^{(2)}}(r_1,\mathbf{x})+g_{\mathbf{m}^{(2)}}(r_2,\mathbf{y}) &= \sum_{t\in S_1}t^2-\sum_{t\in S_2}t^2+\sum_{t\in S_3}t^2. \qquad (2.30)
\end{aligned}$$

According to (2.29) and (2.30), the following linear equation

$$A\boldsymbol{x}=0, \qquad (2.31)$$

where

$$A=\begin{bmatrix}\sum_{t\in S_1}1 & \sum_{t\in S_2}1 & \sum_{t\in S_3}1\\ \sum_{t\in S_1}t & \sum_{t\in S_2}t & \sum_{t\in S_3}t\\ \sum_{t\in S_1}t^2 & \sum_{t\in S_2}t^2 & \sum_{t\in S_3}t^2\end{bmatrix},\quad \boldsymbol{x}=\begin{bmatrix}x_1\\x_2\\x_3\end{bmatrix},$$

has a nonzero solution $(x_1,x_2,x_3)=(1,-1,1)^\top$. We show that this is impossible unless $A=0$. Suppose on the other hand, $A\neq 0$. If all columns of A are not zero columns, then according to the multi-linearity of the determinant,

$$\begin{aligned}
\det(A) &= \sum_{i\in S_1,j\in S_2,k\in S_3}\det\begin{pmatrix}1&1&1\\ i&j&k\\ i^2&j^2&k^2\end{pmatrix}\\
&= \sum_{i\in S_1,j\in S_2,k\in S_3}(j-i)(k-i)(k-j) \qquad (2.32)
\end{aligned}$$

is strictly positive since $\max_{i\in S_1}i<\min_{j\in S_2}j<\min_{k\in S_3}k$. Hence the equation cannot have nonzero solutions. It is also obvious that the equation (2.31) cannot have solution $(x_1,x_2,x_3)=(1,-1,1)^\top$ when only one column A is non-zero. For the case when A contains two non-zero columns, e.g., the first column and the second column, then we have that

$$A'\boldsymbol{x} = \begin{bmatrix} \sum_{t\in S_1} 1 & \sum_{t\in S_2} 1 \\ \sum_{t\in S_1} t & \sum_{t\in S_2} t \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0. \tag{2.33}$$

Again, similar to Eq. (2.32), we have

$$\begin{aligned} \det(A') &= \sum_{i\in S_1, j\in S_2, k\in S_3} \det \begin{pmatrix} 1 & 1 \\ i & j \end{pmatrix} \\ &= \sum_{i\in S_1, j\in S_2} (j-i) > 0, \end{aligned} \tag{2.34}$$

which implies that the equation Eq. (2.33) cannot have nonzero solutions. Thus, Eq. (2.31) cannot have solution $(x_1, x_2, x_3) = (1, -1, 1)^\top$ unless $A = 0$, which implies that S_1, S_2, and S_3 are empty. Therefore, $x_t = 0$ for $t \in \{1, \ldots, s_1+1\}$, $x_t + x_{t+1} = 1$ for $t \in \{s_1+1, \ldots, s_1+s_2-1\}$, and $y_t = 0$ for $t \in \{s_2+1, \ldots, s_2+s_3\}$, which implies (2.11). □

2.4 Protecting $\mathbb{1}_{01}$ Indicator Vectors

We now show that for any $\mathbf{c}$ and $\mathbf{c}'$ in $\{0,1\}^n$ that satisfy $\mathbf{c} \in B_2(\mathbf{c}')$, if $\mathbb{1}_{10}(\mathbf{c}) = \mathbb{1}_{10}(\mathbf{c}')$ and $h(\mathbf{c}) = h(\mathbf{c}')$, then $\mathbb{1}_{01}(\mathbf{c}) = \mathbb{1}_{01}(\mathbf{c}')$. Since $\mathbf{c}$ and $\mathbf{c}'$ have identical 10-indicators, they can be written as

$$\begin{aligned} \mathbf{c} &= 0^{\pi_0}1^{\pi_1}0^{\pi_2}1^{\pi_3}\cdots 0^{\pi_{2\ell}}1^{\pi_{2\ell+1}}, \text{ and} \\ \mathbf{c}' &= 0^{\tau_0}1^{\tau_1}0^{\tau_2}1^{\tau_3}\cdots 0^{\tau_{2\ell}}1^{\tau_{2\ell+1}}, \end{aligned} \tag{2.35}$$

for some integer ℓ, where 0^i (resp. 1^i) denotes a run of i consecutive 0's (resp. 1's), and where $\{\pi_i\}_{i=0}^{2\ell+1}$ and $\{\tau_i\}_{i=0}^{2\ell+1}$ are nonnegative integers such that π_i and τ_i are strictly positive for every $i \notin \{0, 2\ell+1\}$, and such that $\pi_{2i} + \pi_{2i+1} = \tau_{2i} + \tau_{2i+1}$ for all $i \in \{0, 1, \ldots, \ell\}$. In addition, since $h(\mathbf{c})_1 = h(\mathbf{c}')_1$ it follows from Proposition 2.2.2 that $\mathbb{1}_{01}(\mathbf{c}) \cdot \mathbb{1} = \mathbb{1}_{01}(\mathbf{c}') \cdot \mathbb{1}$, i.e., the numbers of 0 runs that is followed by a 1 run in $\mathbf{c}$ and $\mathbf{c}'$ are equal. Note that the numbers of such 0 runs can either be $\ell - 1$, ℓ, and $\ell + 1$, depending on the lengths of the initial and final runs in $\mathbf{c}$ or $\mathbf{c}'$. Hence, we have

$$\begin{aligned} &\mathbb{1}_{01}(\mathbf{c}) \cdot \mathbb{1} = \mathbb{1}_{01}(\mathbf{c}') \cdot \mathbb{1} = \ell + 1 && \text{if } \pi_0 > 0 \text{ and } \pi_{2\ell+1} > 0, \\ &\mathbb{1}_{01}(\mathbf{c}) \cdot \mathbb{1} = \mathbb{1}_{01}(\mathbf{c}') \cdot \mathbb{1} = \ell && \text{if } \pi_0 > 0 \text{ and } \pi_{2\ell+1} = 0 \\ & && \text{or } \pi_0 = 0 \text{ and } \pi_{2\ell+1} > 0, \end{aligned}$$

$$\mathbb{1}_{01}(\mathbf{c}) \cdot \mathbb{1} = \mathbb{1}_{01}(\mathbf{c}') \cdot \mathbb{1} = \ell - 1 \quad \text{if } \pi_0 = 0 \text{ and } \pi_{2\ell+1} = 0. \tag{2.36}$$

Let $\mathbf{d} = 0^{\gamma_0}1^{\gamma_1}0^{\gamma_2}1^{\gamma_3}\cdots 0^{\gamma_{2\ell}}1^{\gamma_{2\ell+1}} \in \{0,1\}^{n-2}$ be a common subsequence of $\mathbf{c}$ and $\mathbf{c}'$ which is obtained by deleting two bits from either $\mathbf{c}$ or $\mathbf{c}'$, where $\gamma_i \geq 0$ for all i. Then, it is readily verified that

$$\sum_{i=0}^{2\ell+1}(\pi_i - \gamma_i) = 2, \ \sum_{i=0}^{2\ell+1}(\tau_i - \gamma_i) = 2, \text{ and hence}$$
$$\sum_{i=1}^{2\ell+1}|\pi_i - \tau_i| \leq \sum_{i=1}^{2\ell+1}|\pi_i - \gamma_i| + \sum_{i=1}^{2\ell+1}|\tau_i - \gamma_i| = 4.$$

Moreover, since $\pi_{2i} + \pi_{2i+1} = \tau_{2i} + \tau_{2i+1}$ for all $i \in \{0,1,\ldots,\ell\}$, it follows that $|\pi_{2i} - \tau_{2i}| = |\pi_{2i+1} - \tau_{2i+1}|$. Since the sum of the (integer) expressions $|\pi_i - \tau_i|$ is at most 4, and since the values of the individual expressions are equal for adjacent values of i (i.e., for $i = 2r$ and $i = 2r+1$ for some integer r), an inequality between the 01-indicators of $\mathbf{c}$ and $\mathbf{c}'$ can only mean one of the following two cases.

Case (a). There exists an integer $j \in [\ell]$ such that $|\pi_{2j} - \tau_{2j}| = |\pi_{2j+1} - \tau_{2j+1}| = 1$ or $|\pi_{2j} - \tau_{2j}| = |\pi_{2j+1} - \tau_{2j+1}| = 2$, and $|\pi_{2i} - \tau_{2i}| = 0$ for $i \neq j$.

Case (b). There exist two integers m and r (where $m < r$) such that $|\pi_{2m} - \tau_{2m}| = |\pi_{2m+1} - \tau_{2m+1}| = 1$ and $|\pi_{2r} - \tau_{2r}| = |\pi_{2r+1} - \tau_{2r+1}| = 1$, and $|\pi_{2i} - \tau_{2i}| = 0$ for $i \notin \{m, r\}$.

In Case (a), since $\pi_{2i} + \pi_{2i+1} = \tau_{2i} + \tau_{2i+1}$ for every i and $\pi_{2i} = \tau_{2i}$ for every $i \neq j$, it follows that $\mathbb{1}_{01}(\mathbf{c})$ and $\mathbb{1}_{01}(\mathbf{c}')$ differ in precisely two positions s and t such that $1 \leq s - t \leq 2$. Hence, since the number of 1's in the 01-indicators is equal, it follows that $\mathbb{1}_{01}(\mathbf{c})_s = \mathbb{1}_{01}(\mathbf{c}')_t$, $\mathbb{1}_{01}(\mathbf{c})_t = \mathbb{1}_{01}(\mathbf{c}')_s$, and $\mathbb{1}_{01}(\mathbf{c})_s \neq \mathbb{1}_{01}(\mathbf{c})_t$, and therefore

$$\begin{aligned} h(\mathbf{c})_2 - h(\mathbf{c}')_2 &= (\mathbb{1}_{01}(\mathbf{c})_s - \mathbb{1}_{01}(\mathbf{c}')_s)\binom{s+1}{2} + \\ &\quad (\mathbb{1}_{01}(\mathbf{c})_t - \mathbb{1}_{01}(\mathbf{c}')_t)\binom{t+1}{2} \\ &= \pm\left(\binom{s+1}{2} - \binom{t+1}{2}\right). \end{aligned} \tag{2.37}$$

Since $1 \leq s - t \leq 2$, it follows that (2.37) equals either $\pm(t+1)$ or $\pm(2t+3)$, and a contradiction follows since neither of which is 0 modulo $2n$, .

Similarly, in Case (b), if none of $\pi_{2m}, \tau_{2m}, \pi_{2m+1}, \tau_{2m+1}, \pi_{2r}, \tau_{2r}, \pi_{2r+1}, \tau_{2r+1}$ is zero, then $\mathbb{1}_{01}(\mathbf{c})$ and $\mathbb{1}_{01}(\mathbf{c}')$ differ in four positions $s, s+1, t$, and $t+1$, and hence

$$\begin{aligned} h(\mathbf{c})_2 - h(\mathbf{c}')_2 = (\mathbb{1}_{01}(\mathbf{c})_s - \mathbb{1}_{01}(\mathbf{c}')_s)\binom{s+1}{2} + \\ (\mathbb{1}_{01}(\mathbf{c})_{s+1} - \mathbb{1}_{01}(\mathbf{c}')_{s+1})\binom{s+2}{2} + \\ (\mathbb{1}_{01}(\mathbf{c})_t - \mathbb{1}_{01}(\mathbf{c}')_t)\binom{t+1}{2} + \\ (\mathbb{1}_{01}(\mathbf{c})_{t+1} - \mathbb{1}_{01}(\mathbf{c}')_{t+1})\binom{t+2}{2}. \end{aligned} \tag{2.38}$$

Once again, since $\mathbb{1}_{01}(\mathbf{c})$ and $\mathbb{1}_{01}(\mathbf{c}')$ have an identical number of 1's, we have that

$$\begin{aligned} \mathbb{1}_{01}(\mathbf{c})_s &= \mathbb{1}_{01}(\mathbf{c}')_{s+1} & \mathbb{1}_{01}(\mathbf{c})_{s+1} &= \mathbb{1}_{01}(\mathbf{c}')_s \\ \mathbb{1}_{01}(\mathbf{c})_t &= \mathbb{1}_{01}(\mathbf{c}')_{t+1} & \mathbb{1}_{01}(\mathbf{c})_{t+1} &= \mathbb{1}_{01}(\mathbf{c}')_t \\ \mathbb{1}_{01}(\mathbf{c})_s &\neq \mathbb{1}_{01}(\mathbf{c}')_s & \mathbb{1}_{01}(\mathbf{c})_t &\neq \mathbb{1}_{01}(\mathbf{c}')_t. \end{aligned}$$

This readily implies that (2.38) equals either $\pm(s-t)$ or $\pm(s+t+2)$, and since none of which is 0 modulo $2n$, another contradiction is obtained. If $\pi_{2m} = 0$ (resp. $\tau_{2m} = 0$), then $\tau_{2m} = 1$ (resp. $\pi_{2m} = 1$). By (2.36) and the discussion after (2.35) it follows that $m = 0$, $r = \ell$, $\tau_{2r+1} = 0$ (resp. $\pi_{2r+1} = 0$), and hence $\mathbb{1}_{01}(\mathbf{c})$ and $\mathbb{1}_{01}(\mathbf{c}')$ differ in the first and last positions. Hence, (2.38) becomes $\pm(1 - \frac{n(n-1)}{2})$, which is nonzero modulo $2n$, and the claim follows.

2.5 Decoding of Two-Deletion Correcting Codes

In this section it is shown how to decode Construction 2.1.1. Recall the encoding function

$$\mathcal{E}(\mathbf{c}) = (\mathbf{c}, f(\mathbf{c}), h(\mathbf{c}), r_3(f(f(\mathbf{c}), h(\mathbf{c}))), r_3(h(f(\mathbf{c}), h(\mathbf{c})))), \tag{2.39}$$

with redundancy $f(\mathbf{c}), h(\mathbf{c})$ of length $N_1 \triangleq 7\log n + 8$ and 3-fold repetition redundancy $r_3(f(f(\mathbf{c}), h(\mathbf{c}))), r_3(h(f(\mathbf{c}), h(\mathbf{c})))$ of length $N_2 \triangleq 21\log(7\log n + 8) + 8$.

To conveniently describe the decoding algorithm, two building blocks are needed. The first is a 3-fold repetition decoding function

$$\mathcal{D}_1 : \{0,1\}^{N_2-2} \to \{0,1\}^{N_2/3}$$

that takes a subsequence $\mathbf{d}_1 \in \{0,1\}^{N_2-2}$ of a 3-fold repetition codeword $r_3(\mathbf{s}_1) \in \{0,1\}^{N_2}$ for some $\mathbf{s}_1 \in \{0,1\}^{N_2/3}$ as input, and outputs an estimate $\tilde{\mathbf{s}}_1$ of the sequence $\mathbf{s}_1$. The second is a decoding function which is defined for every positive integer q as follows

$$\mathcal{D}_2 : \{0,1\}^{q-2} \times \{0,1\}^{7\log q+2} \to \{0,1\}^q.$$

The function $\mathcal{D}_2$ takes a subsequence $\mathbf{d}_2 \in \{0,1\}^{q-2}$ of some $\mathbf{s}_2 \in \{0,1\}^q$, redundancy $f(\mathbf{s}_2)$, and redundancy $h(\mathbf{s}_2)$ as input, and outputs an estimate $\tilde{\mathbf{s}}_2$ of the sequence $\mathbf{s}_2$. In Algorithm 1, the function $\mathcal{D}_2$ is used twice with two different values of q. As will be shown, the two calls of the function $\mathcal{D}_2$ aim to recover the redundancy $f(\mathbf{c})$ and $h(\mathbf{c})$ and the sequence $\mathbf{c}$ respectively.

The 3-fold repetition decoding $\mathcal{D}_1$ can be implemented by adding two bits to $\mathbf{d}_1$ such that the length of each run is a multiple of 3, which can obviously be done in linear time. According to Theorem 2.1.2, there exists a decoding function $\mathcal{D}_2$ that recovers the original sequence $\mathbf{s}_2$ correctly given its $f(\mathbf{s}_2)$ and $h(\mathbf{s}_2)$ redundancy. The linear complexity of $\mathcal{D}_2$ will be shown later in this section.

The functions $\mathcal{D}_1$ and $\mathcal{D}_2$ are used as subroutines to describe the decoding procedure that is given in Algorithm 1. First, we use the function $\mathcal{D}_1$ to recover the redundancy $f(f(\mathbf{c}), h(\mathbf{c}))$ and $h(f(\mathbf{c}), h(\mathbf{c}))$ from the 3-fold repetition code. Then, by applying $\mathcal{D}_2$ and using the redundancy $f(f(\mathbf{c}), h(\mathbf{c}))$ and $h(f(\mathbf{c}), h(\mathbf{c}))$, the $f(\mathbf{c})$ and $h(\mathbf{c})$ can be recovered. Finally and similarly, redundancy $f(\mathbf{c})$ and $h(\mathbf{c})$ can be used to recover the original sequence $\mathbf{c}$, again with the help of $\mathcal{D}_2$.

Algorithm 1: Decoding

```
Input: A subsequence $\mathbf{d} \in \{0,1\}^{N-2}$ of $\mathcal{E}(\mathbf{c})$ for some $\mathbf{c}$ in the code.
Output: The sequence $\mathbf{c}$.
$layer2\_redundancy = \mathcal{D}_1(\mathbf{d}^{(N-N_2+1,N-2)})$;
if two deletions are detected by $\mathcal{D}_1$ then
    return $\mathbf{d}^{(1,n)}$;
else
    $L \triangleq$ The length of the longest suffix of $\mathbf{d}$ that is a subsequence
      of $r_3(layer2\_redundancy)$;
    $layer1\_redundancy = \mathcal{D}_2(\mathbf{d}^{(N-N_1+1-L,N-2-L)}, layer2\_redundancy)$;
    $\mathbf{c} = \mathcal{D}_2(\mathbf{d}^{(1,n-2)}, layer1\_redundancy)$;
    return $\mathbf{c}$.
```

Theorem 2.5.1. *If the functions $\mathcal{D}_1$ and $\mathcal{D}_2$ provide the correct estimates in $O(n)$ time, then given an $N-2$ subsequence of $\mathcal{E}(\mathbf{c})$, Algorithm 1 returns the original sequence $\mathbf{c}$ in $O(n)$ time.*

Proof. To prove the correctness of Algorithm 1, it suffices to show the following

(1). $\mathbf{d}^{(N-N_2+1,N-2)}$ is a length $N_2 - 2$ subsequence of the repetition code $r_3(f(f(\mathbf{c}), h(\mathbf{c}))), r_3(h(f(\mathbf{c}), h(\mathbf{c})))$.

(2). $\mathbf{d}^{(N-N_1+1-L,N-2-L)}$ is a length N_1-2 subsequence of the $f(\mathbf{c}), h(\mathbf{c})$ redundancy.

(3). $\mathbf{d}^{(1,n-2)}$ is a length $n-2$ subsequence of the sequence $\mathbf{c}$.

Since $\mathbf{d}$ is a length $N-2$ subsequence of $\mathcal{E}(\mathbf{c})$, d_{n-2} must be either the $(n-2)$-th, the $(n-1)$-th or the n-th bits of $\mathcal{E}(\mathbf{c})$, and hence (3) must hold. Similarly, (1) holds by considering $\mathbf{d}$ and $\mathcal{E}(\mathbf{c})$ in reversed order. By the definition of L, d_{N-2-L} is the i_1-th bit of $\mathcal{E}(\mathbf{c})$ for some $i_1 \leq n + N_1$. Since (1) holds, we have that L is either N_2, $N_2 - 1$, or $N_2 - 2$. Therefore, d_{N-N_1+1-L} is the i_2-th bit of $\mathcal{E}(\mathbf{c})$ for some $i_2 \geq N - N_1 + 1 - L > n$. Since $(f(\mathbf{c}), h(\mathbf{c})) = \mathcal{E}(\mathbf{c})^{(n+1,n+N_1)}$, (2) must hold.

Since finding L has $O(N_2)$ complexity, the complexity of Algorithm 1 is $O(N) = O(n)$, given that the complexities of the functions $\mathcal{D}_1$ and $\mathcal{D}_2$ are linear. $\square$

It can be verified that Algorithm 1 outputs the original sequence $\mathbf{c}$ in the case of a single deletion. One can also use a VT decoder (see [64]), which has a simpler implementation and $O(n)$ time complexity. We are left to implement $\mathcal{D}_2$ with linear complexity. In particular, we need to recover the sequence $\mathbf{c} \in \{0,1\}^n$ from its length $n-2$ subsequence $\mathbf{d}$ in time $O(n)$, given the redundancies $f(\mathbf{c})$ and $h(\mathbf{c})$. Note that there are $O(n^2)$ supersequences of $\mathbf{d}$ of length n, and f and h can be computed on each of them in $O(n)$. Hence, the brute force approach would require $O(n^3)$.

To achieve linear time complexity, we first recover $\mathbb{1}_{10}(\mathbf{c})$ from an $(n-3)$-subsequence $\mathbb{1}_{10}(\mathbf{d})$ of $\mathbb{1}_{10}(\mathbf{c}) \in \{0,1\}^{n-1}$, and then use it to recover $\mathbf{c}$. In particular, we first find the positions and values of the deleted bits in $\mathbb{1}_{10}(\mathbf{c})$ by an iterative updating algorithm, rather than by exhaustive search, and hence linear complexity is obtained. Furthermore, the uniqueness of the resulting sequence is guaranteed by Lemma 2.2.2. After recovering $\mathbb{1}_{10}(\mathbf{c})$, We can find all length n supersequences $\mathbf{c}'$ of $\mathbf{d}$ such that $\mathbb{1}_{10}(\mathbf{c}') = \mathbb{1}_{10}(\mathbf{c})$. It is shown that there are at most 4 such possible supersequences, and since Theorem 2.1.2 guarantees uniqueness, the right $\mathbf{c}$ is found by computing and comparing $h(\mathbf{c})$. Therefore, the decoding can be done in linear time in total.

Recovering $\mathbb{1}_{10}(\mathbf{c})$.

For $1 \le i \le 2n-2$, let

$$p_i \triangleq \begin{cases} n-i & \text{if } 1 \le i \le n-1 \\ i-n+1 & \text{if } n \le i \le 2n-2 \end{cases}, \text{ and} \tag{2.40}$$

$$b_i \triangleq \begin{cases} 0 & \text{if } 1 \le i \le n-1 \\ 1 & \text{if } n \le i \le 2n-2 \end{cases}. \tag{2.41}$$

For example, when $n = 5$, we have

$$(p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8) = (4, 3, 2, 1, 1, 2, 3, 4) \text{ and}$$
$$(b_1, b_2, b_3, b_4, b_5, b_6, b_7, b_8) = (0, 0, 0, 0, 1, 1, 1, 1).$$

Given a subsequence $\mathbf{d} \in \{0,1\}^{n-2}$ of $\mathbf{c}$, let $\mathbb{1}_{10}(\mathbf{d}) = (r_1, \ldots, r_{n-3})$, and let $w : \{0,1\}^{n-2} \times [2n-2] \times [2n-2] \to \{0,1\}^{n-1} \cup \{\star\}$ be defined as

$$w(\mathbf{d}, i, j) =$$
$$\begin{cases} (r_1, r_2, \ldots, r_{p_i - 1}, b_i, r_{p_i}, \ldots, r_{p_j - 2}, b_j, r_{p_j - 1}, \ldots, r_{n-3}) \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{if } p_i < p_j, \\ (r_1, r_2, \ldots, r_{p_j - 1}, b_j, r_{p_j}, \ldots, r_{p_i - 2}, b_i, r_{p_i - 1}, \ldots, r_{n-3}) \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{if } p_i > p_j, \\ \star \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{if } p_i = p_j, \end{cases}$$

that is, $w(\mathbf{d}, i, j)$ results from $\mathbb{1}_{10}(\mathbf{d})$ inserting b_i at position p_i and b_j in position p_j of $\mathbb{1}_{10}(\mathbf{d})$, if $p_i \ne p_j$. Notice that $w(\mathbf{d}, i, j)$ is one possible candidate for $\mathbb{1}_{10}(\mathbf{c})$. To illustrate $w(\mathbf{d}, i, j)$, let us consider again the example where $n = 5$. Let $\mathbf{d} \in \{0,1\}^{n-2} = (1, 0, 1)$, which implies that $\mathbb{1}_{10}(\mathbf{d}) = (1, 0)$. Then, we have that $w(\mathbf{d}, 2, 7) = \star$ since $p_2 = p_7 = 3$, and that $w(\mathbf{d}, 2, 6) = (1, \underline{1}, \underline{0}, 0)$ since $p_2 = 3, b_2 = 0, p_6 = 2, b_6 = 1$.

For $e \in \{0, 1, 2\}$ define $(2n-2) \times (2n-2)$ integer matrices $\{A^{(e)}\}_{e=0}^{2}$ as follows.

$$A^{(e)}_{i,j} =$$
$$\begin{cases} w(\mathbf{d}, i, j) \cdot \mathbf{m}^{(e)} - \sum_{k=1}^{n-3} \mathbf{m}^{(e)}_k \mathbb{1}_{10}(\mathbf{d})_k & \text{if } w(\mathbf{d}, i, j) \ne \star. \\ \star & \text{if } w(\mathbf{d}, i, j) = \star. \end{cases}$$

Notice that $A^{(e)}_{i,j}$ is the difference in the weighted sums of $w(\mathbf{d}, i, j)$ and $\mathbb{1}_{10}(\mathbf{d})$, with weights $\mathbf{m}^{(e)}$. We now show that the entry of $A^{(e)}$ that we are looking for is equal modulo n_e to the difference in entry e of the f redundancies of $\mathbf{c}$ and of $\mathbf{d}$. That is, $A^{(e)}_{i,j} \equiv (f(\mathbf{c})_e - f(\mathbf{d})_e) \bmod n_e$. On the one hand, since $\mathbb{1}_{10}(\mathbf{c})$ is obtained after two insertions in $\mathbb{1}_{10}(\mathbf{d})$, it follows that there exists (i, j) such that $w(\mathbf{d}, i, j) = \mathbb{1}_{10}(\mathbf{c})$ and that $A^{(e)}_{i,j} \equiv (f(\mathbf{c})_e - f(\mathbf{d})_e) \bmod n_e$ for every $e \in \{0, 1, 2\}$. On the other hand, by Lemma 2.2.2, it follows that this (i, j) pair is unique, given that the sequence $w(\mathbf{d}, i, j)$ does not contain consecutive 1's. Hence, since $w(\mathbf{d}, i, j)$ which contain consecutive 1's are skipped in our algorithm (see Algorithm 2 in the sequel), it follows that $w(\mathbf{d}, i, , j) = \mathbb{1}_{10}(\mathbf{c})$.

We prove the following properties of $A^{(e)}$. In the first property, we give an explicit expression for an entry $A^{(e)}_{i,j}$ in terms of $\mathbb{1}_{10}(\mathbf{d})$, p_i, p_j, b_i, and b_j. This expression will be used for calculating $A^{(e)}_{i,j}$ in constant time from its neighboring entries during $\mathcal{D}_2$. In the following we use $\delta(x)$ to denote the Boolean indicator of an event x, where $\delta(x) = 1$ if and only if x is true.

Proposition 2.5.1. *If $A^{(e)}_{i,j} \neq \star$ then*

$$\begin{aligned} A^{(e)}_{i,j} =& b_i \mathbf{m}^{(e)}_{p_i} + b_j \mathbf{m}^{(e)}_{p_j} + \\ & \sum_{k=1}^{n-3} \mathbb{1}_{10}(\mathbf{d})_k [(k+1)^e \delta(\min\{p_i, p_j\} < k+1) + \\ & (k+2)^e \delta(\max\{p_i, p_j\} < k+2)]. \end{aligned} \tag{2.42}$$

Proof. The difference between $\sum_{k=1}^{n-3} \mathbf{m}^{(e)}_k \mathbb{1}_{10}(\mathbf{d})_k$ and $w(\mathbf{d}, i, j) \cdot \mathbf{m}^{(e)}$ consists of two parts. The first part follows from the two inserted bits, and can be written as

$$b_i \mathbf{m}^{(e)}_{p_i} + b_j \mathbf{m}^{(e)}_{p_j}. \tag{2.43}$$

The second part follows from the shift of bits in $\mathbb{1}_{10}(\mathbf{d})_k$ that is caused by the insertions of two bits b_i and b_j. Each bit $\mathbb{1}_{10}(\mathbf{d})_k$ shifts from position k to position $k+1$ if one insertion occurs before $\mathbb{1}_{10}(\mathbf{d})_k$, i.e., $\min\{p_i, p_j\} < k+1$ and $\max\{p_i, p_j\} \geq k+2$. The resulting difference is given by

$$\begin{aligned} &\sum_{k=1}^{n-3} \mathbb{1}_{10}(\mathbf{d})_k \delta(\min\{p_i, p_j\} < k+1) \cdot \\ &\qquad \delta(\max\{p_i, p_j\} \geq k+2)(\mathbf{m}^{(e)}_{k+1} - \mathbf{m}^{(e)}_k) \\ =& \sum_{k=1}^{n-3} \mathbb{1}_{10}(\mathbf{d})_k \delta(\min\{p_i, p_j\} < k+1) \cdot \end{aligned}$$

$$\delta(\max\{p_i, p_j\} \geq k+2)(k+1)^e. \tag{2.44}$$

The bit $\mathbb{1}_{10}(\mathbf{d})_k$ shifts from position k to $k+2$ if two insertions occur before $\mathbb{1}_{10}(\mathbf{d})_k$, i.e., $\max\{p_i, p_j\} < k+2$. The corresponding difference is given by

$$\begin{aligned}
&\sum_{k=1}^{n-3} \mathbb{1}_{10}(\mathbf{d})_k \delta(\min\{p_i, p_j\} < k+1)\cdot \\
&\qquad \delta(\max\{p_i, p_j\} < k+2)\mathbb{1}_{10}(\mathbf{d})_k(\mathbf{m}_{k+2}^{(e)} - \mathbf{m}_k^{(e)}) \\
&= \sum_{k=1}^{n-3} \mathbb{1}_{10}(\mathbf{d})_k \delta(\max\{p_i, p_j\} < k+2)[(k+1)^e + (k+2)^e].
\end{aligned} \tag{2.45}$$

Combining (2.44) and (2.45), we have that the difference that results from the second part is given by

$$\begin{aligned}
&\sum_{k=1}^{n-3} \mathbb{1}_{10}(\mathbf{d})_k [(k+1)^e \delta(\min\{p_i, p_j\} < k+1) + \\
&\qquad (k+2)^e \delta(\max\{p_i, p_j\} < k+2)],
\end{aligned}$$

that together with (2.43), implies (2.42). □

The following shows that the entries of each $A^{(e)}$ are non-decreasing in rows and columns, and that the respective sequences $w(\mathbf{d}, i, j)$ that lie in the same column or the same row, are unique given each entry value. This property guarantees a simple algorithm for finding a sequence $w(\mathbf{d}, i, j)$ with a given value $A_{i,j}^{(e)}$ by decreasing i or increasing j by 1 in each step.

Proposition 2.5.2. *For every i, j and $i_1 < i_2$, $j_1 < j_2$, if neither of $\mathbf{d}(i_1, j), \mathbf{d}(i_2, j)$, $\mathbf{d}(i, j_1)$, and $\mathbf{d}(i, j_2)$ equals $\star$, then $A_{i_1,j}^{(e)} \leq A_{i_2,j}^{(e)}$ and $A_{i,j_1}^{(e)} \leq A_{i,j_2}^{(e)}$. Moreover, if $A_{i_1,j}^{(e)} = A_{i_2,j}^{(e)}$ (resp. $A_{i,j_1}^{(e)} = A_{i,j_2}^{(e)}$), then $\mathbf{d}(i_1, j) = \mathbf{d}(i_2, j)$ (resp. $\mathbf{d}(i, j_1) = \mathbf{d}(i, j_2)$).*

Proof. By symmetry we only need to prove that the matrix $A^{(e)}$ is non-decreasing in each column, for which it suffices to prove that:

(1). $A_{i_1,j}^{(e)} \leq A_{i_2,j}^{(e)}$ for $1 \leq i_1 < i_2 \leq n-1$.

(2). $A_{n-1,j}^{(e)} \leq A_{n,j}^{(e)}$.

(3). $A_{i_1,j}^{(e)} \leq A_{i_2,j}^{(e)}$ for $n \leq i_1 < i_2 \leq 2n-2$.

For (2), the only difference between $\mathbf{d}(n-1, j)$ and $\mathbf{d}(n, j)$ is that their first bits are 0 and 1 respectively, and hence $A^{(e)}_{n-1,j} + 1 = A^{(e)}_{n,j}$. We are left to show (1) and (3).

(1): For $1 \le i_1 < i_2 \le n-1$, we have $b_{i_1} = b_{i_2} = 0$ and $p_{i_1} > p_{i_2}$. Let $\mathbf{d}'(i_1, j) \in \{0,1\}^{n-2}$ and $\mathbf{d}'(i_2, j) \in \{0,1\}^{n-2}$ be two subsequences of $\mathbf{d}(i_1, j)$ and $\mathbf{d}(i_2, j)$, respectively, after deleting the p_j-th bit from both $\mathbf{d}(i_1, j)$ and $\mathbf{d}(i_2, j)$, and similarly, let $\mathbf{m}^{(e),p_j} = (\mathbf{m}^{(e)}_1, \mathbf{m}^{(e)}_2, \ldots, \mathbf{m}^{(e)}_{p_j-1}, \mathbf{m}^{(e)}_{p_j+1}, \ldots, \mathbf{m}^{(e)}_{n-1})$ be a subsequence of $\mathbf{m}^{(e)}$ after deleting the p_j-th entry. Since $\mathbf{d}'(i_1, j) \in B_1(\mathbf{d}'(i_2, j))$, the remaining arguments follow those in the proof of Proposition 2.1.1. According to (2.6) and (2.7), we have that

$$\begin{aligned} A^{(e)}_{i_2,j} - A^{(e)}_{i_1,j} &= \mathbf{d}(i_2, j) \cdot \mathbf{m}^{(e)} - \mathbf{d}(i_1, j) \cdot \mathbf{m}^{(e)} \\ &= \mathbf{d}'(i_2, j) \cdot \mathbf{m}^{(e),p_j} - \mathbf{d}'(i_1, j) \cdot \mathbf{m}^{(e),p_j} \\ &= g(k_1, \mathbf{d}'(i_2, j)^{(k_1,k_2)}, \mathbf{d}'(i_1, j)_{k_2}) \\ &\ge 0, \end{aligned} \tag{2.46}$$

where $k_1 = p_{i_2} - \delta(p_{i_2} > p_j)$ and $k_2 = p_{i_1} - \delta(p_{i_1} > p_j)$ are the indices whose deletion from $\mathbf{d}'(i_2, j)$ and $\mathbf{d}'(i_1, j)$, respectively, results in $\mathbb{1}_{10}(\mathbf{d})$. Similarly, as in the proof in Proposition 2.1.2, the last inequality follows from the fact that $\mathbf{d}'(i_1, j)_{k_2} = b_{i_1} = 0$. Furthermore, equality holds when $\mathbf{d}'(i_2, j)^{(k_1,k_2)} = 0$ and $\mathbf{d}'(i_1, j)_{k_2} = 0$, which implies that $\mathbf{d}'(i_1, j) = \mathbf{d}'(i_2, j)$, and hence $\mathbf{d}(i_1, j) = \mathbf{d}(i_2, j)$.

(3): For $n \le i_1 < i_2 \le 2n-2$, we have $b_{i_1} = b_{i_2} = 1$ and $p_{i_1} < p_{i_2}$. Similar to (2.46), we have that

$$A^{(e)}_{i_1,j} - A^{(e)}_{i_2,j} = g(k_1, \mathbf{d}'(i_1, j)^{(k_1,k_2)}, \mathbf{d}'(i_2, j)_{k_2}) \le 0,$$

where $k_1 = p_{i_1} - \delta(p_{i_1} > p_j)$ and $k_2 = p_{i_2} - \delta(p_{i_2} > p_j)$ are the indices whose deletion from $\mathbf{d}'(i_1, j)$ and $\mathbf{d}'(i_2, j)$, respectively, results in $\mathbb{1}_{10}(\mathbf{d})$. The last inequality follows from the fact that $\mathbf{d}'(i_2, j)_{k_2} = b_{i_2} = 1$, and equality holds when $\mathbf{d}(i_1, j) = \mathbf{d}(i_2, j)$. □

Remark 2.5.1. *From Proposition 2.5.2, we have that*

$$0 = A^{(e)}_{1,2} \le A^{(e)}_{i,j} \le A^{(e)}_{2n-2,2n-3} \le \mathbf{m}^{(e)}_{n-1} + \mathbf{m}^{(e)}_{n-2} \le n_e$$

for $1 \le i, j \le 2n-2$, $A^{(e)}_{i,j} \ne \star$, *where* $n_0 = 2n$, $n_1 = n^2$, $n_2 = n^3$.

Recall that our goal is to find a sequence $w(\mathbf{d}, i, j) \ne \star$ for which

$$A^{(e)}_{i,j} \equiv f(\mathbf{c})_e - \sum_{i=k}^{n-3} \mathbf{m}^{(e)}_k \mathbb{1}_{10}(\mathbf{d})_k \bmod n_e \tag{2.47}$$

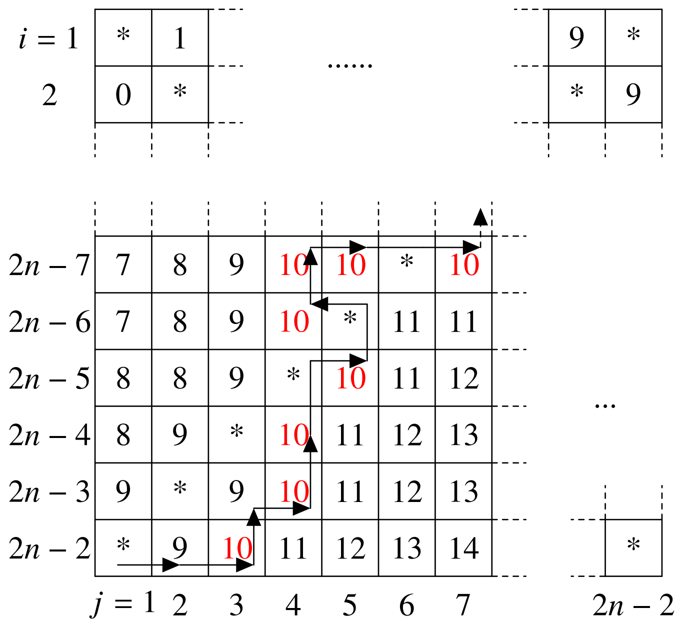

Figure 2.5: The path of Algorithm 2 on the matrix $A^{(0)}$.

for every $e \in \{0,1,2\}$. In addition, the sequence $w(\mathbf{d},i,j)$ cannot contain adjacent 1's, i.e.,

$$w(\mathbf{d},i,j)_{p_i-1}\cdot w(\mathbf{d},i,j)_{p_i} = w(\mathbf{d},i,j)_{p_i}\cdot w(\mathbf{d},i,j)_{p_i+1} = 0$$
$$w(\mathbf{d},i,j)_{p_j-1}\cdot w(\mathbf{d},i,j)_{p_j} = w(\mathbf{d},i,j)_{p_j}\cdot w(\mathbf{d},i,j)_{p_j+1} = 0, \tag{2.48}$$

and from Lemma 2.2.2, such $w(\mathbf{d},i,j)$ equals $\mathbb{1}_{10}(\mathbf{c})$. Moreover, since Remark 2.5.1 implies that $0 \le A^{(e)}_{i,j} \le n_e$, it follows that the modular equality in (2.47) is unnecessary, i.e., it suffices to find a sequence $w(\mathbf{d},i,j) \neq \star$ that satisfies (2.48) and

$$A^{(e)}_{i,j} = a_e \triangleq f_e(\mathbf{c}) - \sum_{k=1}^{n-3} \mathbf{m}^{(e)}_k \mathbb{1}_{10}(\mathbf{d})_k \bmod n_e, \tag{2.49}$$

where a_e is the target value to be found in matrix $A^{(e)}$. Eq. (2.49) implies that $w(\mathbf{d},i,j)$ satisfies the f redundancy.

The procedure to find such $w(\mathbf{d},i,j)$ is given in Algorithm 2. We search for all sequences $w(\mathbf{d},i,j) \neq \star$ with no adjacent 1's (2.48) such that $A^{(0)}_{i,j} = a_0$. This clearly amounts to a binary search in a sorted matrix[3]. We start from the bottom left

[3]The two $\star$ entries in each row or column can simply be skipped.

corner of the matrix, proceed to the right in each step until reaching the rightmost entry such that $A^{(0)}_{i,j} \leq a_0$, and then go one step up. Figure 2.5 illustrates an example of how Algorithm 2 runs on matrix $A^{(0)}$.

To avoid the computation of the entire matrix, that would require $O(n^2)$ time, each entry is computed from previously seen ones *only* upon its discovery. To this end we prove the following lemma, that alongside Proposition 2.5.1, provides a way of computing a newly discovered entry.

Proposition 2.5.3. *Whenever the* (i, j)*-th and* $(i+1, j)$*-th (resp.* $(i, j+1)$*) entries of* $A^{(e)}$ *are not* $\star$*, we have that*

$$\begin{aligned}
&A^{(e)}_{i,j} - A^{(e)}_{i+1,j}\\
= &b_i \mathbf{m}^{(e)}_{p_i} - b_{i+1}\mathbf{m}^{(e)}_{p_{i+1}} + \\
&\sum_{k=\min\{p_i,p_{i+1}\}-1}^{\min\{p_1,p_{i+1}\}} \mathbb{1}_{10}(\mathbf{d})_k[(k+1)^e(\delta(\min\{p_i, p_j\} < k+1)\\
&-\delta(\min\{p_{i+1}, p_j\} < k+1))+\\
&(k+2)^e(\delta(\max\{p_i, p_j\} < k+2)-\\
&\delta(\max\{p_{i+1}, p_j\} < k+2))], \textit{ and}
\end{aligned} \tag{2.50}$$

$$\begin{aligned}
&A^{(e)}_{i,j} - A^{(e)}_{i,j+1}\\
= &b_j \mathbf{m}^{(e)}_{p_j} - b_{j+1}\mathbf{m}^{(e)}_{p_{j+1}} + \\
&\sum_{k=\min\{p_j,p_{j+1}\}-1}^{\min\{p_j,p_{j+1}\}} \mathbb{1}_{10}(\mathbf{d})_k[(k+1)^e(\delta(\min\{p_i, p_j\} < k+1)\\
&-\delta(\min\{p_i, p_{j+1}\} < k+1))+\\
&(k+2)^e(\delta(\max\{p_i, p_j\} < k+2)-\\
&\delta(\max\{p_i, p_{j+1}\} < k+2))].
\end{aligned} \tag{2.51}$$

Proof. Note that if i increases by 1 or if j decreases by 1, then p_i or p_j changes by at most 1 (See (2.40)). Hence,

$$\begin{aligned}
\delta(\min\{p_i, p_j\} < k+1) = \delta(\min\{p_{i+1}, p_j\} < k+1),\\
\delta(\max\{p_i, p_j\} < k+2) = \delta(\max\{p_{i+1}, p_j\} < k+2)
\end{aligned}$$

for $k \leq \min\{p_j, p_{i+1}\} - 2$ and $k \geq \min\{p_i, p_{i+1}\} + 1$. According to (2.42), we have that (2.50) holds, and similarly, (2.51) holds as well. □

Algorithm 2: Finding $\mathbb{1}_{10}(\mathbf{c})$.

```
Input: Subsequence $\mathbf{d} \in \{0,1\}^{n-2}$ of $\mathbf{c}$, and $f(\mathbf{c})$
Output: $i$ and $j$ such that $w(\mathbf{d}, i, j) = \mathbb{1}_{10}(\mathbf{c})$
Initialization: $i = 2n-2$, $j = 1$;
$x_e = A^{(e)}_{1,2n-2}$ for $e \in \{0,1,2\}$;
$a_e = f_e(\mathbf{c}) - \sum_{k=1}^{n-3} \mathbf{m}^{(e)}_k \mathbb{1}_{10}(\mathbf{d})_k \bmod n_e$ for $e \in \{0,1,2\}$;
while $i \geq 0$ do
    if $x_e = a_e$ for every $e \in \{0,1,2\}$ and $w(\mathbf{d}, i, j) \neq \star$ and $w(\mathbf{d}, i, j)$ has no
      adjacent 1's ($w(\mathbf{d}, i, j)$ satisfies (2.48)) then
        return $i, j$;
    else
        Find the maximum $j$ for which $A^{(0)}_{i,j} \leq a_0$.
        if $p_i = p_j$ or ($x_0 > a_0$) then
            $temp\_x_e = x_e + A^{(e)}_{i,j-1} - A^{(e)}_{i,j}$ (using (2.51)), for $e \in \{0,1,2\}$;
            $temp\_j = j - 1$;
            while $p_{temp\_j} = p_i$ do
                $temp\_x_e = x_e + A^{(e)}_{i,temp\_j-1} - A^{(e)}_{i,temp\_j}$ (using (2.51))
                  for $e \in \{0,1,2\}$;
                $temp\_j = temp\_j - 1$;
            if $temp\_j \geq 1$ then
                $j = temp\_j$;
                $x_e = temp\_x_e$ for $e \in \{0,1,2,\}$;
        else
            $temp\_x_e = x_e + A^{(e)}_{i,j+1} - A^{(e)}_{i,j}$ (using (2.51)), for $e \in \{0,1,2\}$;
            $temp\_j = j + 1$;
            while $p_{temp\_j} = p_i$ do
                $temp\_x_e = x_e + A^{(e)}_{i,temp\_j+1} - A^{(e)}_{i,temp\_j}$ (using (2.51))
                  for $e \in \{0,1,2\}$;
                $temp\_j = temp\_j + 1$;
            if $temp\_x_0 \leq a_0$ then
                $j = temp\_j$;
                $x_e = temp\_x_e$ for $e \in \{0,1,2,\}$;
            else
                $x_e = x_e + A^{(e)}_{i-1,j} - A^{(e)}_{i,j}$ (using (2.50));
                $i = i - 1$;
return $(0,0)$;
```

We first show that Algorithm 2 outputs the (i, j) pair such that $w(\mathbf{d}, i, j) = \mathbb{1}_{10}(\mathbf{c})$. Note that by Lemma 2.2.2 there exists a unique sequence $w(\mathbf{d}, i, j) = \mathbb{1}_{10}(\mathbf{c})$ for which $w(\mathbf{d}, i, j)$ satisfies Eq. (2.48) and for which (i, j) satisfies Eq. (2.49). Since the algorithm terminates either when such a sequence $w(\mathbf{d}, i, j) = \mathbb{1}_{10}(\mathbf{c})$ is found or no such sequence is found and i reaches 0, it suffices to show that the latter case does not occur. We prove this by contradiction. Assuming that the latter case occurs, we show that $w(\mathbf{d}, i, j) \neq \mathbb{1}_{10}(\mathbf{c})$ for all (i, j) pairs, which is a contradiction. For each $i \in \{1, 2, \ldots, 2n-2\}$, let j_i be the maximum $j = j_i$ for which $A^{(0)}_{i,j_i} \leq a_0$. If $A^{(0)}_{i,j} > a_0$ for some i and for all j, then $j_i = 1$. Note that each pair (i, j_i) is visited in Algorithm 2 and by assumption we have that $\mathbf{d}(i, j_i) \neq \mathbb{1}_{10}(\mathbf{c})$. We consider the following two cases

(1). $j > j_i$,

(2). $j < j_i$

and conclude that no (i, j) pair in these cases result in $w(\mathbf{d}, i, j) = \mathbb{1}_{10}(\mathbf{c})$. For $j > j_i$, by Proposition 2.5.2 we have that $A^{(0)}_{i,j} \geq A^{(0)}_{i,j_i}$ or that $w(\mathbf{d}, i, j) = \star$. Hence by definition of j_i we have that $A^{(0)}_{i,j} > a_0$ or that $w(\mathbf{d}, i, j) = \star$, and hence $w(\mathbf{d}, i, j) \neq \mathbb{1}_{10}(\mathbf{c})$. For $j < j_i$, by Proposition 2.5.2 we have that $A^{(0)}_{i,j} \leq A^{(0)}_{i,j_i}$ or that $w(\mathbf{d}, i, j) = \star$. If $A^{(0)}_{i,j} < A^{(0)}_{i,j_i}$, then $A^{(0)}_{i,j} \neq a_0$. If $A^{(0)}_{i,j} = A^{(0)}_{i,j_i}$, then according to Proposition 2.5.2, we have that $w(\mathbf{d}, i, j) = \mathbf{d}(i, j_i) \neq \mathbb{1}_{10}(\mathbf{c})$.

We now show that Algorithm 2 terminates in $O(n)$ time. From (2.50) and (2.51) the (i, j)-th entry of $A^{(e)}, e \in \{0, 1, 2\}$, can be computed by using the update rule $x_e + A^{(e)}_{i-1,j} - A^{(e)}_{i,j}$ and $x_e + A^{(e)}_{i,j\pm 1} - A^{(e)}_{i,j}$ (see Algorithm 2), that can be computed in constant time. In addition, one can verify in constant time that (2.48) holds.

Note that in each round, either i decreases by 1 or j increases by 1, with the exception that j decreases by 1 every time when $A^{(0)}_{i,j} = \star$ or $A^{(0)}_{i,j} > a_0$. We prove by contradiction that the latter case, in which $A^{(0)}_{i,j} > a_0$ and $j > 1$ is impossible. Notice that for each current pair (i, j), the value of next pair (i^*, j^*) falls into either one of the following three case:

(1). $(i^*, j^*) = (i, j')$ for some $j' > j$ with $A^{(0)}_{i^*,j^*} \leq a_0$,

(2). $(i^*, j^*) = (i-1, j)$,

(3). $(i^*, j^*) = (i-1, j')$ for some $j' < j$ when $A^{(0)}_{i-1,j} = \star$.

Assume by contradiction that $A^{(0)}_{i^*,j^*} > a_0$ and $j^* > 1$, and (i^*, j^*) is the first visited pair for which this statement is true. In Case (1), we have that $A^{(0)}_{i^*,j^*} \leq a_0$, in contradiction to $A^{(0)}_{i^*,j^*} > a_0$. In Case (2) or Case (3), Proposition 2.5.2 implies that $a_0 < A^{(0)}_{i^*,j^*} \leq A^{(0)}_{i,j}$, contradicting the assumption that (i^*, j^*) is the first visited pair which satisfies $A^{(0)}_{i^*,j^*} > a_0$.

Having proved that $A^{(0)}_{i,j} \leq a_0$ whenever $j > 1$, we have the Algorithm 2 proceeds to the left only when it encounters a $\star$-entry. We now show that the algorithm terminates in $O(n)$ time. Notice that unless Algorithm 2 encounters a $\star$-entry, it proceeds either up or to the right, for which case, it is clear that only $O(n)$ many steps occur. In cases where Algorithm 2 encounters a $\star$-entry, it proceeds to the *left* until a non $\star$-entry is found. Then, this $\star$-entry will not be visited again, because in the next step, it either goes up from the non $\star$-entry or goes to the right of the $\star$-entry. Since the number of $\star$-entries is $4n - 4$, the number of left strides of the algorithm is at most this quantity, and therefore the algorithm terminates in at most $O(n)$ time. In the following, we provide a running example of Algorithm 2.

Example 2.5.1. *Consider a sequence* $\mathbf{c} = (1, 1, 0, 0, 1, 0, 1, 0)$*, where the first and the* 6*-th bits are deleted, resulting in* $\mathbf{d} = (1, 0, 0, 1, 1, 0)$*. Then* $n = 8$*,* $\mathbb{1}_{10}(\mathbf{c}) = (0, 1, 0, 0, 1, 0, 1)$*,* $f(\mathbf{c}) = (14, 46, 200)$*, and* $\mathbb{1}_{10}(\mathbf{d}) = (1, 0, 0, 0, 1)$*. Hence* $a_0 = 8, a_1 = 30, a_2 = 144$.

Then, Algorithm 2 proceeds in the following manner. The underlined bits denote the inserted bits to $\mathbb{1}_{10}(\mathbf{d})$.

$$\begin{aligned}
&i = 1, j = 14, p_i = p_j, x_0 = 7, x_1 = 28, x_2 = 140\\
\to &i = 2, j = 14, w(\mathbf{d}, i, j) = (1, 0, 0, 0, 1, \underline{0}, \underline{1}),\\
&x_0 = 7, x_1 = 28, x_2 = 140\\
\to &i = 3, j = 14, w(\mathbf{d}, i, j) = (1, 0, 0, 0, \underline{0}, 1, \underline{1}),\\
&x_0 = 8, x_1 = 34, x_2 = 176,\\
\to &i = 4, j = 14, w(\mathbf{d}, i, j) = (1, 0, 0, \underline{0}, 0, 1, \underline{1}),\\
&x_0 = 8, x_1 = 34, x_2 = 176,\\
\to &i = 5, j = 14, w(\mathbf{d}, i, j) = (1, 0, \underline{0}, 0, 0, 1, \underline{1}),\\
&x_0 = 8, x_1 = 34, x_2 = 176,\\
\to &i = 6, j = 14, w(\mathbf{d}, i, j) = (1, \underline{0}, 0, 0, 0, 1, \underline{1}),\\
&x_0 = 8, x_1 = 34, x_2 = 176,
\end{aligned}$$

$$\rightarrow i = 7, j = 14, w(\mathbf{d}, i, j) = (\underline{0}, 1, 0, 0, 0, 1, \underline{1}),$$
$$x_0 = 9, x_1 = 36, x_2 = 180$$
$$\rightarrow i = 7, j = 13, w(\mathbf{d}, i, j) = (\underline{0}, 1, 0, 0, 0, \underline{1}, 1),$$
$$x_0 = 9, x_1 = 36, x_2 = 180$$
$$\rightarrow i = 7, j = 12, w(\mathbf{d}, i, j) = (\underline{0}, 1, 0, 0, \underline{1}, 0, 1),$$
$$x_0 = 8, x_1 = 30, x_2 = 144$$

Recover the original sequence c

Let (i, j) be the output of Algorithm 2, for which we have that $w(\mathbf{d}, i, j) = \mathbb{1}_{10}(\mathbf{c})$. Let $\mathbf{c}'$ be a length n supersequence after two insertions to $\mathbf{d}$ such that $\mathbb{1}_{10}(\mathbf{c}') = \mathbb{1}_{10}(\mathbf{c})$. If $b_i = 1$, then inserting b_i to $\mathbb{1}_{10}(\mathbf{d})$ corresponds to either inserting a 0 to $\mathbf{d}$ as the p_i+1-th bit in $\mathbf{c}'$ or inserting a 1 to $\mathbf{d}$ as the p_i-th bit in $\mathbf{c}'$ (see Table 2.1). If $b_i = 0$, then inserting b_i to $\mathbb{1}_{10}(\mathbf{d})$ corresponds to inserting a 0 or 1 in the first 0 run or 1 run respectively after the k'-th bit in $\mathbf{c}'$, where $k' = \max_k\{w(\mathbf{d}, i, j)_k = 1, k < p_i\}$ is the index of the last 10-pattern that occurs before the p_i-th bit in $\mathbf{c}'$. The same arguments hold for the insertion of b_j.

Therefore, given the (i, j) pair that Algorithm 2 returns, there are at most four possible $\mathbf{c}'$ supersequences of $\mathbf{d}$ such that $\mathbb{1}_{10}(\mathbf{c}') = \mathbb{1}_{10}(\mathbf{c})$. One can check if the $\mathbf{c}'$ sequences satisfy $h(\mathbf{c})$. According to Theorem 2.1.2, there is a unique such sequence, the original sequence $\mathbf{c}$ that satisfies both $f(\mathbf{c})$ and $h(\mathbf{c})$ simultaneously.

2.6 Appendix

Proof of (2.13) (Case (a))

$$\begin{aligned}
&(\mathbb{1}_{10}(\mathbf{c}) - \mathbb{1}_{10}(\mathbf{c}')) \cdot \mathbf{m}^{(e)} \\
=&\sum_{t=\ell_1}^{\ell_2} (\mathbb{1}_{10}(\mathbf{c})_t - \mathbb{1}_{10}(\mathbf{c}')_t) \cdot (\mathbf{m}^{(e)})_t + \\
&\sum_{t=k_2}^{k_1} (\mathbb{1}_{10}(\mathbf{c})_t - \mathbb{1}_{10}(\mathbf{c}')_t) \cdot (\mathbf{m}^{(e)})_t \\
=&(\mathbb{1}_{10}(\mathbf{c})_{\ell_2} - \mathbb{1}_{10}(\mathbf{c}')_{\ell_2}) \cdot (\mathbf{m}^{(e)})_{\ell_2} + \\
&(\mathbb{1}_{10}(\mathbf{c})_{k_1} - \mathbb{1}_{10}(\mathbf{c}')_{k_1}) \cdot (\mathbf{m}^{(e)})_{k_1} + \\
&\sum_{t=\ell_1}^{\ell_2-1} (\mathbb{1}_{10}(\mathbf{c})_t - \mathbb{1}_{10}(\mathbf{c})_{t+1}) \cdot (\mathbf{m}^{(e)})_t +
\end{aligned}$$

$$\begin{aligned}
&\sum_{t=k_2}^{k_1-1} (\mathbb{1}_{10}(\mathbf{c}')_{t+1} - \mathbb{1}_{10}(\mathbf{c}')_t) \cdot (\mathbf{m}^{(e)})_t \\
&= (\mathbb{1}_{10}(\mathbf{c})_{\ell_2} - \mathbb{1}_{10}(\mathbf{c}')_{\ell_2}) \cdot (\mathbf{m}^{(e)})_{\ell_2} + \\
&\quad (\mathbb{1}_{10}(\mathbf{c})_{k_1} - \mathbb{1}_{10}(\mathbf{c}')_{k_1}) \cdot (\mathbf{m}^{(e)})_{k_1} + \\
&\quad \sum_{t=\ell_1}^{\ell_2-1} \mathbb{1}_{10}(\mathbf{c})_t \cdot (\mathbf{m}^{(e)})_t - \sum_{\ell_1+1}^{\ell_2} \mathbb{1}_{10}(\mathbf{c})_t \cdot (\mathbf{m}^{(e)})_{t-1} \\
&\quad + \sum_{t=k_2+1}^{k_1} \mathbb{1}_{10}(\mathbf{c}')_t \cdot (\mathbf{m}^{(e)})_{t-1} - \sum_{t=k_2}^{k_1-1} \mathbb{1}_{10}(\mathbf{c}')_t \cdot (\mathbf{m}^{(e)})_t \\
&= (\mathbb{1}_{10}(\mathbf{c})_{\ell_2} - \mathbb{1}_{10}(\mathbf{c}')_{\ell_2}) \cdot (\mathbf{m}^{(e)})_{\ell_2} + \\
&\quad (\mathbb{1}_{10}(\mathbf{c})_{k_1} - \mathbb{1}_{10}(\mathbf{c}')_{k_1}) \cdot (\mathbf{m}^{(e)})_{k_1} + \\
&\quad \mathbb{1}_{10}(\mathbf{c})_{\ell_1} \cdot (\mathbf{m}^{(e)})_{\ell_1} - \mathbb{1}_{10}(\mathbf{c})_{\ell_2} \cdot (\mathbf{m}^{(e)})_{\ell_2-1} + \\
&\quad \sum_{t=\ell_1+1}^{\ell_2-1} \mathbb{1}_{10}(\mathbf{c})_t \cdot t^e + \mathbb{1}_{10}(\mathbf{c}')_{k_1} \cdot (\mathbf{m}^{(e)})_{k_1-1} \\
&\quad - \mathbb{1}_{10}(\mathbf{c}')_{k_2} \cdot (\mathbf{m}^{(e)})_{k_2} - \sum_{t=k_2+1}^{k_1-1} \mathbb{1}_{10}(\mathbf{c}')_t \cdot t^e \\
&= (-\mathbb{1}_{10}(\mathbf{c}')_{\ell_2}) \cdot (\mathbf{m}^{(e)})_{\ell_2} + (\mathbb{1}_{10}(\mathbf{c})_{k_1}) \cdot (\mathbf{m}^{(e)})_{k_1} + \\
&\quad \mathbb{1}_{10}(\mathbf{c})_{\ell_1} \cdot (\mathbf{m}^{(e)})_{\ell_1} + \sum_{t=\ell_1+1}^{\ell_2} \mathbb{1}_{10}(\mathbf{c})_t \cdot t^e - \\
&\quad \mathbb{1}_{10}(\mathbf{c}')_{k_2} \cdot (\mathbf{m}^{(e)})_{k_2} - \sum_{t=k_2+1}^{k_1} \mathbb{1}_{10}(\mathbf{c}')_t \cdot t^e \\
&= \mathbb{1}_{10}(\mathbf{c})_{\ell_1} \cdot (\mathbf{m}^{(e)})_{\ell_1} + \mathbb{1}_{10}(\mathbf{c})_{k_1} \cdot (\mathbf{m}^{(e)})_{k_1} + \\
&\quad \sum_{t=\ell_1+1}^{\ell_2} \mathbb{1}_{10}(\mathbf{c})_t \cdot t^e - \sum_{t=k_2+1}^{k_1} \mathbb{1}_{10}(\mathbf{c}')_t \cdot t^e - \\
&\quad \left(\mathbb{1}_{10}(\mathbf{c}')_{\ell_2} \cdot (\mathbf{m}^{(e)})_{\ell_2} + \mathbb{1}_{10}(\mathbf{c}')_{k_2} \cdot (\mathbf{m}^{(e)})_{k_2}\right) \\
&= g_{\mathbf{m}^{(e)},\ell_1}(\mathbb{1}_{10}(\mathbf{c})_{\ell_1}, \ldots, \mathbb{1}_{10}(\mathbf{c})_{\ell_2}, \mathbb{1}_{10}(\mathbf{c}')_{\ell_2}) - \\
&\quad g_{\mathbf{m}^{(e)},k_2}(\mathbb{1}_{10}(\mathbf{c}')_{k_2}, \ldots, \mathbb{1}_{10}(\mathbf{c}')_{k_1}, \mathbb{1}_{10}(\mathbf{c})_{k_1})
\end{aligned}$$

Proof of (2.14) (Case (b))

$$(\mathbb{1}_{10}(\mathbf{c}) - \mathbb{1}_{10}(\mathbf{c}')) \cdot \mathbf{m}^{(e)}$$

$$
\begin{aligned}
&= \sum_{t=\ell_1}^{\ell_2} \left(\mathbb{1}_{10}(\mathbf{c})_t - \mathbb{1}_{10}(\mathbf{c}')_t\right) \cdot (\mathbf{m}^{(e)})_t + \\
&\quad \sum_{t=k_1}^{k_2} \left(\mathbb{1}_{10}(\mathbf{c})_t - \mathbb{1}_{10}(\mathbf{c}')_t\right) \cdot (\mathbf{m}^{(e)})_t \\
&= \left(\mathbb{1}_{10}(\mathbf{c})_{\ell_2} - \mathbb{1}_{10}(\mathbf{c}')_{\ell_2}\right) \cdot (\mathbf{m}^{(e)})_{\ell_2} + \\
&\quad \left(\mathbb{1}_{10}(\mathbf{c})_{k_2} - \mathbb{1}_{10}(\mathbf{c}')_{k_2}\right) \cdot (\mathbf{m}^{(e)})_{k_2} + \\
&\quad \sum_{t=\ell_1}^{\ell_2-1} \left(\mathbb{1}_{10}(\mathbf{c})_t - \mathbb{1}_{10}(\mathbf{c})_{t+1}\right) \cdot (\mathbf{m}^{(e)})_t + \\
&\quad \sum_{t=k_1}^{k_2-1} \left(\mathbb{1}_{10}(\mathbf{c})_t - \mathbb{1}_{10}(\mathbf{c})_{t+1}\right) \cdot (\mathbf{m}^{(e)})_t \\
&= \left(\mathbb{1}_{10}(\mathbf{c})_{\ell_2} - \mathbb{1}_{10}(\mathbf{c}')_{\ell_2}\right) \cdot (\mathbf{m}^{(e)})_{\ell_2} + \\
&\quad \left(\mathbb{1}_{10}(\mathbf{c})_{k_2} - \mathbb{1}_{10}(\mathbf{c}')_{k_2}\right) \cdot (\mathbf{m}^{(e)})_{k_2} + \\
&\quad \sum_{t=\ell_1}^{\ell_2-1} \mathbb{1}_{10}(\mathbf{c})_t \cdot (\mathbf{m}^{(e)})_t - \sum_{\ell_1+1}^{\ell_2} \mathbb{1}_{10}(\mathbf{c})_t \cdot (\mathbf{m}^{(e)})_{t-1} + \\
&\quad \sum_{t=k_1}^{k_2-1} \mathbb{1}_{10}(\mathbf{c})_t \cdot (\mathbf{m}^{(e)})_t - \sum_{t=k_1+1}^{k_2} \mathbb{1}_{10}(\mathbf{c})_t \cdot (\mathbf{m}^{(e)})_{t-1} \\
&= \left(\mathbb{1}_{10}(\mathbf{c})_{\ell_2} - \mathbb{1}_{10}(\mathbf{c}')_{\ell_2}\right) \cdot (\mathbf{m}^{(e)})_{\ell_2} + \\
&\quad \left(\mathbb{1}_{10}(\mathbf{c})_{k_2} - \mathbb{1}_{10}(\mathbf{c}')_{k_2}\right) \cdot (\mathbf{m}^{(e)})_{k_2} + \\
&\quad \mathbb{1}_{10}(\mathbf{c})_{\ell_1} \cdot (\mathbf{m}^{(e)})_{\ell_1} - \mathbb{1}_{10}(\mathbf{c})_{\ell_2} \cdot (\mathbf{m}^{(e)})_{\ell_2-1} + \\
&\quad \sum_{t=\ell_1+1}^{\ell_2-1} \mathbb{1}_{10}(\mathbf{c})_t \cdot t^e + \mathbb{1}_{10}(\mathbf{c})_{k_1} \cdot (\mathbf{m}^{(e)})_{k_1} \\
&\quad - \mathbb{1}_{10}(\mathbf{c})_{k_2} \cdot (\mathbf{m}^{(e)})_{k_2-1} + \sum_{t=k_1+1}^{k_2-1} \mathbb{1}_{10}(\mathbf{c})_t \cdot t^e \\
&= \left(-\mathbb{1}_{10}(\mathbf{c}')_{\ell_2}\right) \cdot (\mathbf{m}^{(e)})_{\ell_2} + \left(-\mathbb{1}_{10}(\mathbf{c}')_{k_2}\right) \cdot (\mathbf{m}^{(e)})_{k_2} + \\
&\quad \mathbb{1}_{10}(\mathbf{c})_{\ell_1} \cdot (\mathbf{m}^{(e)})_{\ell_1} + \sum_{t=\ell_1+1}^{\ell_2} \mathbb{1}_{10}(\mathbf{c})_t \cdot t^e + \\
&\quad \mathbb{1}_{10}(\mathbf{c})_{k_1} \cdot (\mathbf{m}^{(e)})_{k_1} + \sum_{t=k_1+1}^{k_2} \mathbb{1}_{10}(\mathbf{c})_t \cdot t^e \\
&= \mathbb{1}_{10}(\mathbf{c})_{\ell_1} \cdot (\mathbf{m}^{(e)})_{\ell_1} + \mathbb{1}_{10}(\mathbf{c})_{k_1} \cdot (\mathbf{m}^{(e)})_{k_1} - \\
&\quad \left(\mathbb{1}_{10}(\mathbf{c}')_{\ell_2} \cdot (\mathbf{m}^{(e)})_{\ell_2} + \mathbb{1}_{10}(\mathbf{c}')_{k_2} \cdot (\mathbf{m}^{(e)})_{k_2}\right) +
\end{aligned}
$$

$$\sum_{t=\ell_1+1}^{\ell_2} \mathbb{1}_{10}(\mathbf{c})_t \cdot t^e + \sum_{t=k_1+1}^{k_2} \mathbb{1}_{10}(\mathbf{c})_t \cdot t^e$$
$$= g_{\mathbf{m}^{(e)},\ell_1}(\mathbb{1}_{10}(\mathbf{c})_{\ell_1}, \ldots, \mathbb{1}_{10}(\mathbf{c})_{\ell_2}, \mathbb{1}_{10}(\mathbf{c}')_{\ell_2}) +$$
$$g_{\mathbf{m}^{(e)},k_1}(\mathbb{1}_{10}(\mathbf{c})_{k_1}, \ldots, \mathbb{1}_{10}(\mathbf{c})_{k_2}, \mathbb{1}_{10}(\mathbf{c}')_{k_2})$$

Proof of (2.15) (Case (c))

$$(\mathbb{1}_{10}(\mathbf{c}) - \mathbb{1}_{10}(\mathbf{c}')) \cdot \mathbf{m}^{(e)}$$
$$= \sum_{t=\ell_1}^{k_1-2} (\mathbb{1}_{10}(\mathbf{c})_t - \mathbb{1}_{10}(\mathbf{c}')_t) \cdot (\mathbf{m}^{(e)})_t +$$
$$\sum_{t=k_1-1}^{\ell_2-1} (\mathbb{1}_{10}(\mathbf{c})_t - \mathbb{1}_{10}(\mathbf{c}')_t) \cdot (\mathbf{m}^{(e)})_t +$$
$$\sum_{t=\ell_2}^{k_2} (\mathbb{1}_{10}(\mathbf{c})_t - \mathbb{1}_{10}(\mathbf{c}')_t) \cdot (\mathbf{m}^{(e)})_t$$
$$= \sum_{t=\ell_1}^{k_1-2} (\mathbb{1}_{10}(\mathbf{c})_t - \mathbb{1}_{10}(\mathbf{c})_{t+1}) \cdot (\mathbf{m}^{(e)})_t +$$
$$\sum_{t=k_1-1}^{\ell_2-1} (\mathbb{1}_{10}(\mathbf{c})_t - \mathbb{1}_{10}(\mathbf{c})_{t+2}) \cdot (\mathbf{m}^{(e)})_t +$$
$$(\mathbb{1}_{10}(\mathbf{c})_{\ell_2} - \mathbb{1}_{10}(\mathbf{c}')_{\ell_2}) \cdot (\mathbf{m}^{(e)})_{\ell_2} +$$
$$(\mathbb{1}_{10}(\mathbf{c})_{k_2} - \mathbb{1}_{10}(\mathbf{c}')_{k_2}) \cdot (\mathbf{m}^{(e)})_{k_2} +$$
$$\sum_{t=\ell_2+1}^{k_2-1} (\mathbb{1}_{10}(\mathbf{c})_t - \mathbb{1}_{10}(\mathbf{c})_{t+1}) \cdot (\mathbf{m}^{(e)})_t$$
$$= \sum_{t=\ell_1}^{k_1-2} \mathbb{1}_{10}(\mathbf{c})_t \cdot (\mathbf{m}^{(e)})_t - \sum_{t=\ell_1+1}^{k_1-1} \mathbb{1}_{10}(\mathbf{c})_t \cdot (\mathbf{m}^{(e)})_{t-1} +$$
$$\sum_{t=k_1-1}^{\ell_2} \mathbb{1}_{10}(\mathbf{c})_t \cdot (\mathbf{m}^{(e)})_t - \sum_{t=k_1+1}^{\ell_2+1} \mathbb{1}_{10}(\mathbf{c})_t \cdot (\mathbf{m}^{(e)})_{t-2} +$$
$$(-\mathbb{1}_{10}(\mathbf{c}')_{\ell_2}) \cdot (\mathbf{m}^{(e)})_{\ell_2} + (-\mathbb{1}_{10}(\mathbf{c}')_{k_2}) \cdot (\mathbf{m}^{(e)})_{k_2} +$$
$$\sum_{t=\ell_2+1}^{k_2} \mathbb{1}_{10}(\mathbf{c})_t \cdot (\mathbf{m}^{(e)})_t - \sum_{t=\ell_2+2}^{k_2} \mathbb{1}_{10}(\mathbf{c})_t \cdot (\mathbf{m}^{(e)})_{t-1}$$
$$= \mathbb{1}_{10}(\mathbf{c})_{\ell_1}(\mathbf{m}^{(e)})_{\ell_1} - \mathbb{1}_{10}(\mathbf{c})_{k_1-1}(\mathbf{m}^{(e)})_{k_1-2} +$$

$$\sum_{t=\ell_1+1}^{k_1-2} \mathbb{1}_{10}(\mathbf{c})_t \cdot t^e + \mathbb{1}_{10}(\mathbf{c})_{k_1-1}(\mathbf{m}^{(e)})_{k_1-1} +$$
$$\mathbb{1}_{10}(\mathbf{c})_{k_1}(\mathbf{m}^{(e)})_{k_1} - \mathbb{1}_{10}(\mathbf{c})_{\ell_2+1}(\mathbf{m}^{(e)})_{\ell_2-1} +$$
$$\sum_{t=k_1+1}^{\ell_2} \mathbb{1}_{10}(\mathbf{c})_t(t^e + (t-1)^e) + (-\mathbb{1}_{10}(\mathbf{c}')_{\ell_2}) \cdot (\mathbf{m}^{(e)})_{\ell_2} +$$
$$(-\mathbb{1}_{10}(\mathbf{c}')_{k_2}) \cdot (\mathbf{m}^{(e)})_{k_2} + \mathbb{1}_{10}(\mathbf{c})_{\ell_2+1}(\mathbf{m}^{(e)})_{\ell_2+1} +$$
$$\sum_{t=\ell_2+2}^{k_2} \mathbb{1}_{10}(\mathbf{c})_t t^e$$
$$= \mathbb{1}_{10}(\mathbf{c})_{\ell_1}(\mathbf{m}^{(e)})_{\ell_1} + \mathbb{1}_{10}(\mathbf{c})_{k_1}(\mathbf{m}^{(e)})_{k_1} -$$
$$(\mathbb{1}_{10}(\mathbf{c}')_{\ell_2} \cdot (\mathbf{m}^{(e)})_{\ell_2} + \mathbb{1}_{10}(\mathbf{c}')_{k_2} \cdot (\mathbf{m}^{(e)})_{k_2}) +$$
$$\sum_{t=\ell_1+1}^{k_1-1} \mathbb{1}_{10}(\mathbf{c})_t \cdot t^e + \sum_{t=k_1+1}^{\ell_2+1} \mathbb{1}_{10}(\mathbf{c})_t(t^e + (t-1)^e) +$$
$$\sum_{t=\ell_2+2}^{k_2} \mathbb{1}_{10}(\mathbf{c})_t t^e$$
$$= \mathbb{1}_{10}(\mathbf{c})_{\ell_1}(\mathbf{m}^{(e)})_{\ell_1} + \mathbb{1}_{10}(\mathbf{c})_{k_1}(\mathbf{m}^{(e)})_{k_1} -$$
$$(\mathbb{1}_{10}(\mathbf{c}')_{\ell_2} \cdot (\mathbf{m}^{(e)})_{\ell_2} + \mathbb{1}_{10}(\mathbf{c}')_{k_2} \cdot (\mathbf{m}^{(e)})_{k_2}) +$$
$$\sum_{t=\ell_1+1}^{k_1-1} \mathbb{1}_{10}(\mathbf{c})_t \cdot t^e + \sum_{t=k_1}^{\ell_2} \mathbb{1}_{10}(\mathbf{c})_{t+1} t^e + \sum_{t=k_1+1}^{k_2} \mathbb{1}_{10}(\mathbf{c})_t t^e$$
$$= g_{\mathbf{m}^{(e)},\ell_1}(\mathbb{1}_{10}(\mathbf{c})_{\ell_1}, \ldots, \mathbb{1}_{10}(\mathbf{c})_{k_1-1}, \mathbb{1}_{10}(\mathbf{c})_{k_1+1}, \ldots,$$
$$\mathbb{1}_{10}(\mathbf{c})_{\ell_2+1}, \mathbb{1}_{10}(\mathbf{c}')_{\ell_2}) +$$
$$g_{\mathbf{m}^{(e)},k_1}(\mathbb{1}_{10}(\mathbf{c})_{k_1}, \ldots, \mathbb{1}_{10}(\mathbf{c})_{k_2}, \mathbb{1}_{10}(\mathbf{c}')_{k_2})$$

Chapter 3

BINARY CODES CORRECTING k DELETIONS/INSERTIONS

Following the construction in Ch. 2, this chapter presents binary codes correcting any constant number of deletion/inssertion errors.

3.1 Introduction

The problem of constructing efficient k-deletion codes (defined in Ch. 2) has long been unsettled. Before [12], which proposed a k-deletion correcting code construction with $O(k^2 \log k \log n)$ redundancy, no deletion codes correcting more than a single error with rate approaching 1 was known.

After [12], the work in [44] considered correction with high probability and proposed a k-deletion correcting code construction with redundancy $(k+1)(2k+1)\log n + o(\log n)$ and encoding/decoding complexity $O(n^{k+1}/\log^{k-1} n)$. The result for this randomized coding setting was improved in [41], where redundancy $O(k\log(n/k))$ and complexity $poly(n,k)$ were achieved. However, finding a deterministic k-deletion correcting code construction that achieves the optimal order redundancy $O(k \log n)$ remained elusive. Note that the optimality of the code is redundancy-wise rather than cardinality-wise, namely, the focus is on the asymptotic rather than exact size of the code.

In this chapter, we provide a solution to this longstanding open problem for constant k: We present a code construction that achieves $O(8k\log n + o(\log n))$ redundancy when $k = o(\sqrt{\log\log n})$ and $O(n^{2k+1})$ encoding/decoding computational complexity. Note that the complexity is polynomial in n when k is a constant. The following theorem summarizes our main result.

Theorem 3.1.1. *Let k and n be two integers satisfying $k = o(\sqrt{\log\log n})$. For integer $N = n + 8k\log n + o(\log n)$, there exists an encoding function $\mathcal{E} : \{0,1\}^n \to \{0,1\}^N$, computed in $O(n^{2k+1})$ time, and a decoding function $\mathcal{D} : \{0,1\}^{N-k} \to \{0,1\}^n$, computed in $O(n^{k+1}) = O(N^{k+1})$ time, such that for any $\mathbf{c} \in \{0,1\}^n$ and subsequence $\mathbf{d} \in \{0,1\}^{N-k}$ of $\mathcal{E}(\mathbf{c})$, we have that $\mathcal{D}(\mathbf{d}) = \mathbf{c}$.*

An independent work [22] proposed a k-deletion correcting code with $O(k\log n)$ redundancy and better complexity of $poly(n,k)$. In contrast to redundancy $8k\log n$

in our construction, the redundancy in [22] is not specified, and it is estimated to be at least $200k \log n$. Moreover, the techniques in [22] and in our constructions are different. Our techniques can be applied to obtain a systematic k-deletion code construction with $4k \log n + o(\log n)$ bits of redundancy, as well as deletion codes for other settings, which will be given in the next chapter.

Here are the key building blocks in our code construction: (i) *generalizing the VT construction* to k deletions by considering constrained sequences, (ii) separating the encoded vector to blocks and using *concatenated codes* and (iii) a novel strategy to *separate the vector to blocks by a single pattern.* The following gives a more specific description of these ideas.

In the previous chapter, we *generalized the VT construction*. In particular, we proved that while the higher order parity checks $\sum_{i=1}^{N} i^j c_i \bmod (N^j + 1)$, $j = 0, 1, \ldots, t$ do not work in general, those parity checks work in the two-deletion case, when the sequences are constrained to have no consecutive 1's. In this chapter we generalize this idea, specifically, the higher order parity checks can be used to correct $k = t/2$ deletions in sequences that satisfy the *following constraint:* The index distance between any two 1's is at least k, i.e., there is a 0 run of length at least $k - 1$ between any two 1's.

The fact that we can correct k deletions using the generalization of the VT construction on constrained sequences, enables a *concatenated code construction*, which is the underlying structure of our k-deletion correcting codes. In concatenated codes, each codeword $\mathbf{c}$ is split into blocks of small length, by certain markers at the boundaries between adjacent blocks. Each block is protected by an inner deletion code that can be efficiently computed when the block length is small. The block boundary markers are chosen such that k deletions in a codeword result in at most $O(k)$ block errors. Then, it suffices to use an outer code, such as a Reed-Solomon code, to correct the block errors.

Concatenated codes were used in a number of k-deletion correcting code constructions, [12, 40, 81]. One of the main differences among these constructions is the choice of the block boundary markers that separate the codewords. The constructions in [81, 40] insert extra bits as markers between blocks, which introduces $O(N)$ bits of redundancy. In [12], occurrences of short subsequences in the codeword, called patterns, were used as markers. The construction in [12] requires $O(k \log k)$ different patterns where each pattern introduces $O(k \log N)$ bits of redundancy.

We improve the redundancy in [12] by using *a single type of pattern* for block boundary markers. Specifically, we choose a pattern such that its indicator vector, a binary vector in which the 1 entries indicate the occurrences of the pattern in $\mathbf{c}$, is a *constrained sequence* that is immune to deletions. Then, the generalized VT construction can be used to protect the indicator vector and thus the locations of the block boundary markers. Knowing the boundary between blocks, we can recover the blocks with at most $2k$ block errors, which then can be corrected using a combination of short deletion correcting codes and Reed-Solomon codes.

The concatenated code construction consists of short blocks, hence, the pattern has to occur frequently in the codeword such that all consecutive bits of certain length contains at least one occurrence of the pattern. Namely, the first step of the encoding is a mapping of an arbitrary binary sequence to a sequence with frequent synchronization patterns. The constructions in [12, 22] compute such mappings using randomized algorithms. In this chapter, we provide a deterministic algorithm to generate sequences with frequent synchronization patterns.

We now formally define the synchronization pattern and its indicator vector.

Definition 3.1.1. *A* synchronization pattern*, is a length* $3k+\lceil\log k\rceil+4$ *sequence* $\mathbf{a} = (a_1, \dots, a_{3k+\lceil\log k\rceil+4})$ *satisfying*

- $a_{3k+i} = 1$ *for* $i \in [0, \lceil\log k\rceil+4]$*, where* $[0, \lceil\log k\rceil+4] = \{0, \dots, \lceil\log k\rceil+4\}$.
- *There does not exist a* $j \in [1, 3k-1]$*, such that* $a_{j+i} = 1$ *for* $i \in [0, \lceil\log k\rceil+4]$.

Namely, a synchronization pattern is a length $3k + \lceil\log k\rceil + 4$ *sequence that ends with* $\lceil\log k\rceil + 5$ *consecutive* 1*'s and no other 1-run with length* $\lceil\log k\rceil + 5$ *exists.*

For a sequence $\mathbf{c} = (c_1, \dots, c_n)$, the indicator vector of the synchronization pattern, referred to as *synchronization vector* $\mathbb{1}_{sync}(\mathbf{c}) \in \{0, 1\}^n$, is defined by

$$\mathbb{1}_{sync}(\mathbf{c})_i = \begin{cases} 1, & \text{if } (c_{i-3k+1}, c_{i-3k+2}, \dots, c_{i+\lceil\log k\rceil+4}) \\ & \text{is a synchronization pattern,} \\ 0, & \text{else.} \end{cases} \tag{3.1}$$

Note that $\mathbb{1}_{sync}(\mathbf{c})_i = 0$ for $i \in [1, 3k-1]$ and for $i \in [n - \lceil\log k\rceil - 3, n]$. The entry $\mathbb{1}_{sync}(\mathbf{c}) = 1$ if and only if c_i is the first bit of the final 1 run in a synchronization pattern. It can be seen from the definition that any two 1 entries in $\mathbb{1}_{sync}(\mathbf{c})$ have index distance at least $3k$, i.e., any two 1 entries are separated by a 0-run of length at least $3k-1$. Hence, $\mathbb{1}_{sync}(\mathbf{c})$ is a constrained sequence described above.

Example 3.1.1. *Let integers $k=2$ and $n=35$. Then the sequence* $(1,0,0,1,0,1,1,1,1,1,1)$ *is a synchronization pattern. Let the length n sequence*

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

Then the synchronization vector $\mathbb{1}_{sync}(\mathbf{c})$ is given by

$$\begin{aligned}\mathbb{1}_{sync}(\mathbf{c}) = (&0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,\\&0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0).\end{aligned}$$

Next, we present the definition of the generalized VT code. Define the integer vectors

$$\mathbf{m}^{(e)} \triangleq (1^e, 1^e+2^e, \ldots, \sum_{j=1}^{n} j^e) \tag{3.2}$$

for $e \in [0, 6k]$, where the i-th entry $\mathbf{m}_i^{(e)}$ of $\mathbf{m}^{(e)}$ is the sum of the e-th powers of the first i positive integers. Given a sequence $\mathbf{c} \in \{0,1\}^n$, we compute the generalized VT redundancy $f(\mathbf{c})$ of dimension $6k+1$ as follows:

$$\begin{aligned}f(\mathbf{c})_e &\triangleq \mathbf{c} \cdot \mathbf{m}^{(e)} \bmod 3kn^{e+1}\\&= \sum_{i=1}^{n} c_i \mathbf{m}_i^{(e)} \bmod 3kn^{e+1},\end{aligned} \tag{3.3}$$

for $e \in [0, 6k]$, where $f(\mathbf{c})_e$ is the e-th component of $f(\mathbf{c})$ and $\cdot$ denotes inner product over integers. Our generalization of the VT code construction shows that the vector $f(\mathbb{1}_{sync}(\mathbf{c}))$ helps protect the synchronization vector $\mathbb{1}_{sync}(\mathbf{c})$ from k deletions in $\mathbf{c}$.

The rest of the chapter is organized as follows. Sec. 3.2 provides an outline of our construction and some of the basic lemmas. Based on results of the basic lemmas, Sec. 3.3 presents the encoding and decoding procedures of our code. Sec. 3.4 presents our VT generalization for recovering the synchronization vector. Sec. 3.5 explains how to correct k deletions based on the synchronization vector, when the synchronization patterns appear frequently. Sec. 3.6 describes an algorithm to transform a sequence into one with frequently occurred synchronization patterns. Sec. 3.7 concludes this chapter.

3.2 Outline and Preliminaries

In this section, we outline the ingredients that constitute our code construction and present notations that will be used throughout this chapter. A summary of these notations is provided in Table 3.1. We begin with an overview of the code construction, which has a concatenated code structure as described in Sec. 3.1. In our construction, each codeword $\mathbf{c}$ is split into blocks, with the boundaries between adjacent blocks given by synchronization patterns. Specifically, let $t_1, \ldots, t_J$ be the indices of the synchronization patterns in $\mathbf{c}$, i.e., the indices of the 1 entries in $\mathbb{1}_{sync}(\mathbf{c})$. Then the blocks are given by $(c_{t_{j-1}+1}, \ldots, c_{t_j-1})$ for $j \in [0, J+1]$, where $t_j = 0$ if $j = 0$ and $t_j = n+1$ if $j = J+1$. The key idea of our construction is to use the VT generalization f (see Eq. (3.3) for definition) as parity checks to protect the synchronization vector $\mathbb{1}_{\mathbf{c}}$, and thus identify the indices of block boundaries. The proof details will be given in Lemma 3.2.1. Note that the parity $f(\mathbf{c})$ in (3.3) has size $O(k^2 \log n)$. To compress the size of $f(\mathbf{c})$ to the targeted $O(k \log n)$, in Lemma 3.2.2 we apply a modulo operation on the function f. Given the block boundaries, Lemma 3.2.3 provides an algorithm to protect the codewords. Specifically, we show that given the synchronization vector $\mathbb{1}_{sync}(\mathbf{c})$, it is possible to recover most of the blocks in $\mathbf{c}$, with up to $2k$ block errors. These block errors can be corrected with the help of their deletion correcting hashes, which can be exhaustively computed and will be presented in Lemma 8.2.3. Hence, to correct the block errors, it suffices to protect the sequence of deletion correcting hashes using, for example, Reed-Solomon codes.

Lemma 3.2.2 and Lemma 3.2.3 together define a hash function that protects $\mathbf{c}$ from k deletions. However, in order to exhaustively compute the block hash given in Lemma 8.2.3 and obtain the desired size of the redundancy, the block length has to be of order $O(poly(k) \log n)$. Hence the synchronization pattern must appear frequently enough in $\mathbf{c}$. Such sequence $\mathbf{c}$ will be defined in the following as a k-dense sequence. To encode for any given sequence $\mathbf{c} \in \{0, 1\}^n$, in Lemma 3.2.4 we present an invertible mapping that takes any sequence $\mathbf{c} \in \{0, 1\}^n$ as input and outputs a k-dense sequence. Then, any binary sequence can be encoded into a k-dense sequence and protected.

Finally, the hash function defined by Lemma 3.2.2 and Lemma 3.2.3 is subject to deletion errors and has to be protected. To this end, we use an additional hash that encodes the deletion correcting hash of the hash function defined in Lemma 3.2.2 and Lemma 3.2.3. The additional hash is protected by a $(k+1)$-fold repetition code. Similar additional hash technique was also given in [12].

Table 3.1: Summary of Notations in Ch. 3

$\mathcal{B}_k(\mathbf{c})$	$\triangleq$	The set of sequences that share a length $n-k$ subsequence with $\mathbf{c} \in \{0,1\}^n$.
$\mathcal{R}_m$	$\triangleq$	The set of sequences where any two 1 entries are separated by at least $m-1$ zeros.
$\mathbb{1}_{sync}(\mathbf{c})$	$\triangleq$	Synchronization vector defined in (3.1).
$\mathbf{m}^{(e)}$	$\triangleq$	Weights of order e in the VT generalization, defined in (3.2).
$f(\mathbf{c})$	$\triangleq$	Generalized VT redundancy defined in (3.3).
L	$\triangleq$	Maximal length of zero runs in the synchronization vector of a k-*dense* sequence, defined in (3.4).
$p(\mathbf{c})$	$\triangleq$	The function used to compute the redundancy protecting the synchronization vector $\mathbb{1}_{sync}(\mathbf{c})$, defined in Lemma 3.2.2.
$Hash_k(\mathbf{c})$	$\triangleq$	The deletion correcting hash function for k-dense sequences, defined in Lemma 3.2.3.
$T(\mathbf{c})$	$\triangleq$	The function that generates k-dense sequences, defined in Lemma 3.2.4.
$H(\mathbf{c})$	$\triangleq$	Deletion correcting hash function for any sequence, defined in Lemma 8.2.3.

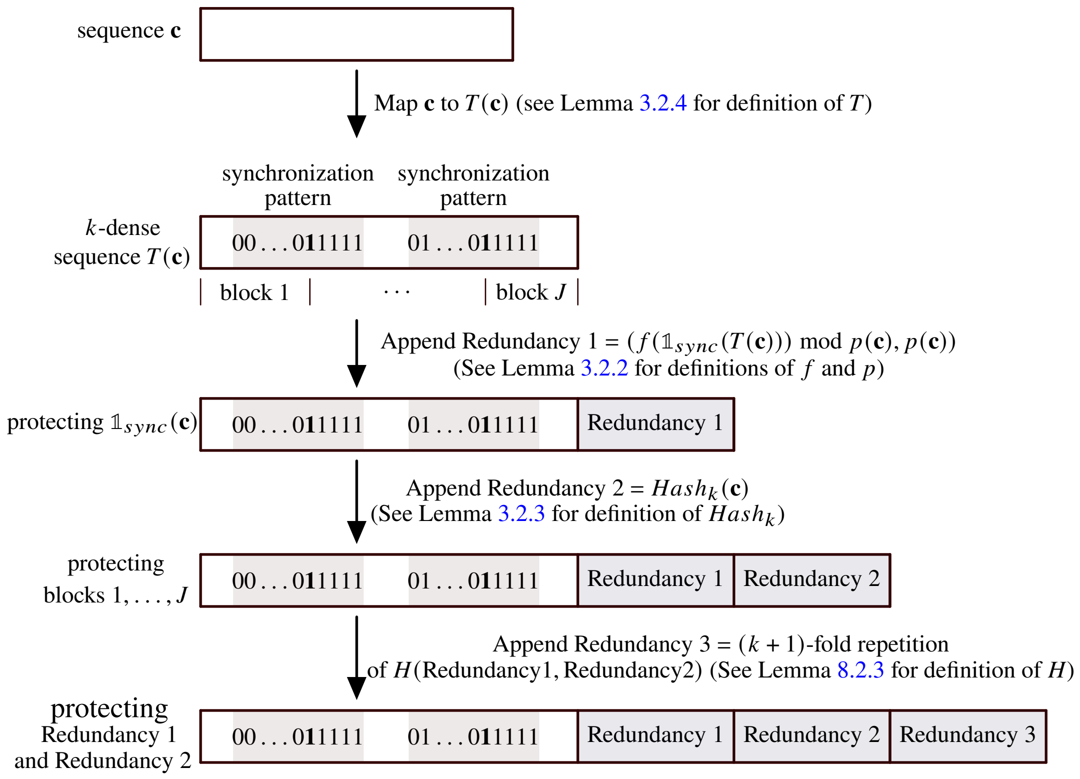
sequence $\mathbf{c}$
Map $\mathbf{c}$ to $T(\mathbf{c})$ (see Lemma 3.2.4 for definition of T)
synchronization pattern
synchronization pattern
k-dense sequence $T(\mathbf{c})$
$00\ldots011111$
$01\ldots011111$
block 1
$\cdots$
block J
Append Redundancy 1 = $(f(\mathbb{1}_{sync}(T(\mathbf{c}))) \bmod p(\mathbf{c}), p(\mathbf{c}))$
(See Lemma 3.2.2 for definitions of f and p)
protecting $\mathbb{1}_{sync}(\mathbf{c})$
Redundancy 1
Append Redundancy 2 = $Hash_k(\mathbf{c})$
(See Lemma 3.2.3 for definition of $Hash_k$)
protecting blocks $1,\ldots,J$
Redundancy 2
Append Redundancy 3 = $(k+1)$-fold repetition
of H(Redundancy1, Redundancy2) (See Lemma 8.2.3 for definition of H)
protecting Redundancy 1 and Redundancy 2
Redundancy 3

Figure 3.1: Illustrating the encoding procedure of our k-deletion correcting code construction.

The final encoding/decoding algorithm of our code, which combines the results in Lemma 3.2.2, Lemma 3.2.3, and Lemma 3.2.4, is provided in Sec. 3.3. The encoding is illustrated in Fig. 3.1.

Before presenting the lemmas, we give necessary definitions and notations. For a sequence $\mathbf{c} \in \{0,1\}^n$, define its deletion ball $\mathcal{B}_k(\mathbf{c})$ as the collection of sequences that share a length $n-k$ subsequence with $\mathbf{c}$.

Definition 3.2.1. *A sequence $\mathbf{c} \in \{0,1\}^n$ is said to be k-*dense *if the lengths of the* 0 *runs in $\mathbb{1}_{sync}(\mathbf{c})$ is at most*

$$L \triangleq (\lceil \log k \rceil + 5)2^{\lceil \log k \rceil + 9}\lceil \log n \rceil + (3k + \lceil \log k \rceil + 4)(\lceil \log n \rceil + 9 + \lceil \log k \rceil). \tag{3.4}$$

For k-dense $\mathbf{c}$, the index distance between any two 1 *entries in $\mathbb{1}_{sync}(\mathbf{c})$ is at most $L+1$, i.e., the* 0*-runs between two* 1 *entries have length at most $L+1$.*

Note that $f(\mathbf{c})$ is an integer vector that can be presented by $\log(3k)^{6k+1}n^{(3k+1)(6k+1)}$ bits or by an integer in the range $[0,(3k)^{6k+1}n^{(3k+1)(6k+1)}-1]$. In this chapter, we interchangeably use $f(\mathbf{c})$ to denote its binary presentation or integer presentation. The following lemma shows that the synchronization vector $\mathbb{1}_{sync}(\mathbf{c})$ can be recovered from k deletions with the help of $f(\mathbb{1}_{sync}(\mathbf{c}))$. Its proof will be given in Sec. 3.4.

Lemma 3.2.1. *For integers n and k and sequences $\mathbf{c}, \mathbf{c}'$, if $\mathbf{c}' \in \mathcal{B}_k(\mathbf{c})$ and $f(\mathbb{1}_{sync}(\mathbf{c})) = f(\mathbb{1}_{sync}(\mathbf{c}'))$, then $\mathbb{1}_{sync}(\mathbf{c}) = \mathbb{1}_{sync}(\mathbf{c}')$.*

By virtue of Lemma 3.2.1, the synchronization vector $\mathbb{1}_{sync}(\mathbf{c})$ can be recovered from k deletions in $\mathbf{c}$, with the help of a hash $f(\mathbb{1}_{sync}(\mathbf{c}))$ of size $O(k^2 \log n)$ bits. To further reduce the size of the hash to $O(k \log n)$ bits, we apply modulo operations on $f(\mathbb{1}_{sync}(\mathbf{c}))$ in the following lemma, the proof of which will be proved in Sec. 3.4.

Lemma 3.2.2. *For integers n and $k = o(\sqrt{\log \log n})$, there exists a function $p : \{0,1\}^n \to [1, 2^{2k \log n + o(\log n)}]$, such that if $f(\mathbb{1}_{sync}(\mathbf{c})) \equiv f(\mathbb{1}_{sync}(\mathbf{c}')) \bmod p(\mathbf{c})$ for two sequences $\mathbf{c} \in \{0,1\}^n$ and $\mathbf{c}' \in \mathcal{B}_k(\mathbf{c})$, then $\mathbb{1}_{sync}(\mathbf{c}) = \mathbb{1}_{sync}(\mathbf{c}')$. Hence if*

$$\begin{aligned}&(f(\mathbb{1}_{sync}(\mathbf{c})) \bmod p(\mathbf{c}), p(\mathbf{c}))\\=&(f(\mathbb{1}_{sync}(\mathbf{c}')) \bmod p(\mathbf{c}'), p(\mathbf{c}'))\end{aligned}$$

and $\mathbf{c}' \in \mathcal{B}_k(\mathbf{c})$, we have that $\mathbb{1}_{sync}(\mathbf{c}) = \mathbb{1}_{sync}(\mathbf{c}')$.

Lemma 3.2.2 presents a hash of size $4k \log n + o(\log n)$ bits for correcting $\mathbb{1}_{sync}(\mathbf{c})$. With the knowledge of the synchronization vector $\mathbb{1}_{sync}(\mathbf{c})$, the next lemma shows that the sequence $\mathbf{c}$ can be further recovered using another $4k \log + o(\log n)$ bit hash, when $\mathbf{c}$ is k-dense, i.e., when the synchronization patter occurs frequently enough in $\mathbf{c}$. The proof of Lemma 3.2.3 will be given in Sec. 3.5.

Lemma 3.2.3. *For integers n and $k = o(\sqrt{\log \log n})$, there exists a function $Hash_k : \{0,1\}^n \to \{0,1\}^{4k \log n + o(\log n)}$, such that every k-dense sequence $\mathbf{c} \in \{0,1\}^n$ can be recovered, given its synchronization vector $\mathbb{1}_{sync}(\mathbf{c})$, its length $n-k$ subsequence $\mathbf{d}$, and $Hash_k(\mathbf{c})$.*

Combining Lemma 3.2.2 and Lemma 3.2.3, we obtain size $O(k \log n)$ hash function to correct deletions for a k-dense sequence. To encode for arbitrary sequence $\mathbf{c} \in \{0,1\}^n$, a mapping that transforms any sequence to a k-dense sequence is given in the following lemma. The details will be given in Sec. 3.6.

Lemma 3.2.4. *For integers k and $n > k$, there exists a map $T : \{0,1\}^n \to \{0,1\}^{n+3k+3\lceil \log k \rceil+15}$, computable in $poly(n,k)$ time, such that $T(\mathbf{c})$ is a k-dense sequence for $\mathbf{c} \in \{0,1\}^n$. Moreover, the sequence $\mathbf{c}$ can be recovered from $T(\mathbf{c})$.*

The next three lemmas (Lemma 8.2.3, Lemma 3.2.6, and Lemma 3.2.7) present existence results that are necessary to prove Lemma 3.2.2 and Lemma 3.2.3. Lemma 8.2.3 gives a k-deletion correcting hash function for short sequences, which is the block hash described above in proving Lemma 3.2.3. It is an extension of the result in [12]. Lemma 3.2.6 is a slight variation of the result in [64]. It shows the equivalence between correcting deletions and correcting deletions and insertions. Lemma 3.2.6 will be used in the proof of Lemma 3.2.1, where we need an upper bound on the number of deletions/insertions in $\mathbb{1}_{sync}(\mathbf{c})$ caused by a deletion in $\mathbf{c}$. Lemma 3.2.7 (see [71]) gives an upper bound on the number of divisors of a positive integer n. With Lemma 3.2.7, we show that the VT generalization in Lemma 3.2.1 can be compressed by taking modulo operations. The details will be given in the proof of Lemma 3.2.2.

Lemma 3.2.5. *For any integers w, n, and k, there exists a hash function $H : \{0,1\}^w \to$
$\{0,1\}^{\lceil (w/\lceil \log n \rceil) \rceil (2k \log\log n + O(1))}$, computable in $O_k((w/\log n) n \log^{2k} n)$ time, such that any sequence $\mathbf{c} \in \{0,1\}^w$ can be recovered from its length $w-k$ subsequence $\mathbf{d}$ and the hash $H(\mathbf{c})$.*

Proof. We first show by counting arguments the existence of a hash function $H' : \{0,1\}^{\lceil \log n \rceil} \to \{0,1\}^{2k \log\log n + O(1)}$, exhaustively computable in $O_k(n \log^{2k} n)$ time, such that $H'(\mathbf{s}) \neq H'(\mathbf{s}')$ for all $\mathbf{s} \in \{0,1\}^{\lceil \log n \rceil}$ and $\mathbf{s}' \in \mathcal{B}_k(\mathbf{s}) \backslash \{\mathbf{s}\}$. The hash $H'(\mathbf{c}')$ protects the sequence $\mathbf{s} \in \{0,1\}^{\lceil \log n \rceil}$ from k deletions. Note that $|\mathcal{B}_k(\mathbf{c}')| \leq \binom{\lceil \log n \rceil}{k}^2 2^k \leq 2\lceil \log n \rceil^{2k}$. Hence it suffices to use brute force and greedily assign a hash value for each sequence $\mathbf{s} \in \{0,1\}^{\lceil \log n \rceil}$ such that $H'(s) \neq H'(\mathbf{s}')$ for all $\mathbf{s}' \in \mathcal{B}_k(\mathbf{s}) \backslash \{\mathbf{s}\}$. Since the size of $\mathcal{B}_k(\mathbf{s})$ is upper bounded by $2\lceil \log n \rceil^{2k}$, there always exists such a hash $H'(\mathbf{s}) \in \{0,1\}^{\log(2\lceil \log n \rceil^{2k}+1)}$, that has no conflict with the hash values of sequences in $\mathcal{B}_k(\mathbf{s})$. The total complexity is $O_k(n \log^{2k} n)$ and the size of the hash value $H'(\mathbf{s})$ is $2k \log\log n + O(1)$.

Now split $\mathbf{c}$ into $\lceil (w/\lceil \log n \rceil) \rceil$ blocks $c_{(i-1)\lceil \log n \rceil+1}, \ldots, c_{i\lceil \log n \rceil}$,
$i \in [1, \lceil (w/\lceil \log n \rceil) \rceil]$ of length $\lceil \log n \rceil$. If the length of the last block is less than $\lceil \log n \rceil$, add zeros to the end of the last block such that its length is $\lceil \log n \rceil$. Assign a hash value $\mathbf{h}_i = H'((c_{(i-1)\lceil \log n \rceil+1}, \ldots, c_{i\lceil \log n \rceil}))$, $i \in [1, \lceil (w/\lceil \log n \rceil) \rceil]$

for each block. Let $H(\mathbf{c}) = (\mathbf{h}_1, \ldots, \mathbf{h}_{\lceil (w/\lceil \log n \rceil) \rceil})$ be the concatenation of $\mathbf{h}_i$ for $i \in [1, \lceil (w/\lceil \log n \rceil) \rceil]$.

We show that $H(\mathbf{c})$ protects $\mathbf{c}$ from k deletions. Let $\mathbf{d}$ be a length $n-k$ subsequence of $\mathbf{c}$. Note that $d_{(i-1)\lceil \log n \rceil+1}$ and $d_{i\lceil \log n \rceil-k}$ come from bits $c_{(i-1)\lceil \log n \rceil+1+x}$ and $c_{i\lceil \log n \rceil-k+y}$ respectively after deletions in $\mathbf{c}$, where the integers $x, y \in [0, k]$. Therefore, $(d_{(i-1)\lceil \log n \rceil+1}, \ldots, d_{i\lceil \log n \rceil-k})$ is a length $\lceil \log n \rceil - k$ subsequence of $(c_{(i-1)\lceil \log n \rceil+1+x}, \ldots, c_{i\lceil \log n \rceil-k+y})$, and thus a subsequence of the block $(c_{(i-1)\lceil \log n \rceil+1}, \ldots, c_{i\lceil \log n \rceil})$. Hence the i-th block $(c_{(i-1)\lceil \log n \rceil+1}, \ldots, c_{i\lceil \log n \rceil})$ can be recovered from $\mathbf{h}_i = H'(c_{(i-1)\lceil \log n \rceil+1}, \ldots, c_{i\lceil \log n \rceil})$ and $(d_{(i-1)\lceil \log n \rceil+1}, \ldots, d_{i\lceil \log n \rceil-k})$. Therefore, $\mathbf{c}$ can be recovered given $\mathbf{d}$ and $H(\mathbf{c})$. The length of $H(\mathbf{c})$ is $\lceil (w/\lceil \log n \rceil) \rceil \cdot (2k \log \log n + O(1))$ and the complexity of $H(\mathbf{c})$ is $O_k((w/\log n) n \log^{2k} n)$. □

Lemma 3.2.6. *Let r, s, and k be integers satisfying $r + s \leq k$. For sequences $\mathbf{c}, \mathbf{c}' \in \{0,1\}^n$, if $\mathbf{c}'$ and $\mathbf{c}$ share a common resulting sequence after r deletions and s insertions in both, then $\mathbf{c}' \in \mathcal{B}_k(\mathbf{c})$.*

Lemma 3.2.7. *For a positive integer $n \geq 3$, the number of divisors of n is upper bounded by $2^{1.6 \ln n / (\ln \ln n)}$.*

The proofs of Lemma 3.2.1, Lemma 3.2.2, Lemma 3.2.3, and Lemma 3.2.4 rely on several propositions, the details of which will be presented in the next sections. For convenience, a dependency graph for the theorem, lemmas, and propositions is given in Fig. 3.2.

3.3 Proof of Theorem 3.1.1

Based on the lemmas stated in Sec. 3.2, in this section we present the encoding function $\mathcal{E}$ and the decoding function $\mathcal{D}$ of our k-deletion correcting code, and prove Theorem 3.1.1. Given any sequence $\mathbf{c} \in \{0,1\}^n$, let the function $\mathcal{E} : \{0,1\}^n \to \{0,1\}^{n+8k\log n + o(\log n)}$ be given by

$$\mathcal{E}(\mathbf{c}) = (T(\mathbf{c}), R'(\mathbf{c}), R''(\mathbf{c})),$$

where

$$\begin{aligned} R'(\mathbf{c}) =& (f(\mathbb{1}_{sync}(T(\mathbf{c}))) \bmod p(T(\mathbf{c})), p(T(\mathbf{c})), Hash_k(T(\mathbf{c}))), \text{ and} \\ R''(\mathbf{c}) =& Rep_{k+1}(H(R'(\mathbf{c}))). \end{aligned}$$

Function $Rep_{k+1}(H(R'(\mathbf{c})))$ is the $(k+1)$-fold repetition of the bits in $H(R'(\mathbf{c}))$, where function $H(R'(\mathbf{c}))$ is defined Lemma 8.2.3 and protects $R'(\mathbf{c})$ from k dele-

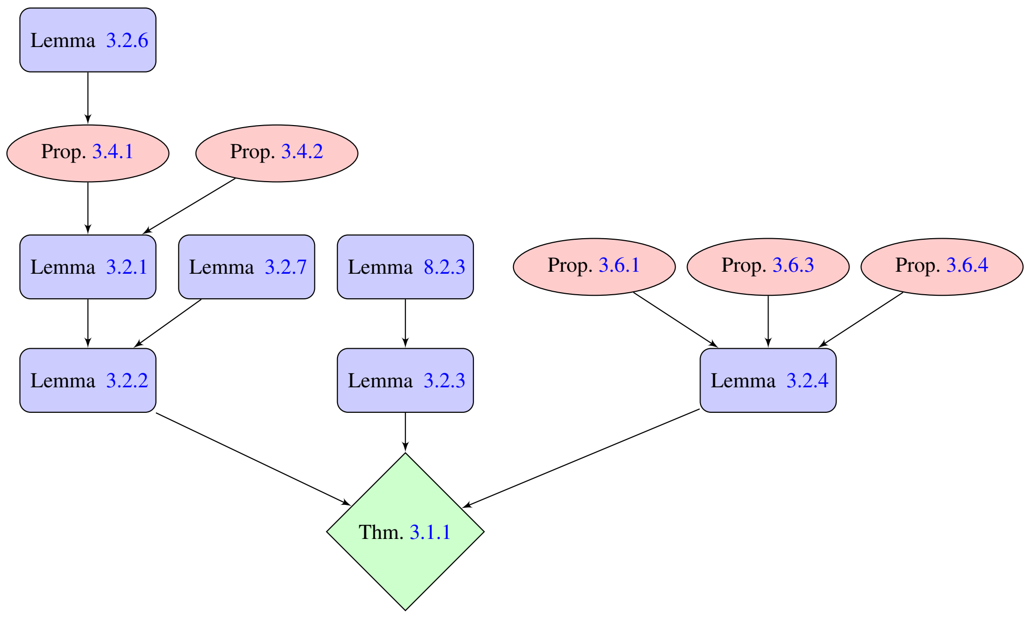

Figure 3.2: Dependencies of the claims in Ch. 3.

tions. Function $T(\mathbf{c})$ is defined in Lemma 3.2.4 and transforms $\mathbf{c}$ into a k-dense sequence (see Definition 3.2.1 for definition of a k-dense sequence). Function $f(\mathbb{1}_{sync}(T(\mathbf{c})))$ in $R'(\mathbf{c})$ is represented by an integer and protects the synchronization vector $\mathbb{1}_{sync}(\mathbf{c})$ by Lemma 3.2.1. Function $p(T(\mathbf{c}))$ is defined in Lemma 3.2.2, which compresses the hash $f(\mathbb{1}_{sync}(T(\mathbf{c})))$. Function $Hash_k(T(\mathbf{c}))$ is defined in Lemma 3.2.3 and protects a k-dense sequence from k deletions.

Note that $k = o(\sqrt{\log\log n})$. Hence, according to Lemma 3.2.2, Lemma 3.2.3, and Lemma 3.2.4, the length of $R'(\mathbf{c})$ is $N_1 = 8k\log(n+3k+3\lceil\log k\rceil+15) + o(\log(n+3k+3\lceil\log k\rceil+15)) = 8k\log n + o(\log n)$. The length of $R''(\mathbf{c})$ is $N_2 = 2k(k+1)(N_1/\lceil\log n\rceil)\log\log n = o(\log n)$. The length of $T(\mathbf{c})$ is $n+N_0 = n+3k+3\lceil\log k\rceil+15$. Therefore, the length of $\mathcal{E}(\mathbf{c})$ is $n+N_0+N_1+N_2 = n+8k\log n+o(\log n)$. The redundancy of the code is $8k\log n + o(\log n)$.

To show how the sequence $\mathbf{c}$ can be recovered from a length $N-k$ subsequence $\mathbf{d}$ of $\mathcal{E}(\mathbf{c})$ and implement the computation of the decoding function $\mathcal{D}(\mathbf{d})$, we prove that

1. **Statement 1:** The redundancy $R'(\mathbf{c})$ can be recovered given $(d_{n+N_0+N_1+1}, \ldots, d_{n+N_0+N_1+N_2-k})$.

2. **Statement 2:** The sequence $\mathbf{c}$ can be recovered given $(d_1, \ldots, d_{n+N_0-k})$ and $R'(\mathbf{c})$.

We first prove Statement 1. Note that $d_{n++N_0+N_1+1}$ and $d_{n+N_0+N_1+N_2-k}$ come from bits $\mathcal{E}(\mathbf{c})_{n+N_0+N_1+1+x}$ and $\mathcal{E}(\mathbf{c})_{n+N_0+N_1+N_2-k+y}$ respectively after deletions in $\mathcal{E}(\mathbf{c})$, where $x, y \in [0, k]$. Hence $(d_{n+N_0+N_1+1}, \ldots, d_{n+N_0+N_1+N_2-k})$ is a length $N_2 - k$ subsequence of $(\mathcal{E}(\mathbf{c})_{n+N_0+N_1+1+x}, \ldots, \mathcal{E}(\mathbf{c})_{n+N_0+N_1+N_2-k+y})$, and thus a subsequence of $(\mathcal{E}(\mathbf{c})_{n+N_0+N_1+1}, \ldots, \mathcal{E}(\mathbf{c})_{n+N_0+N_1+N_2}) = R''(\mathbf{c})$. Since $R''(\mathbf{c})$ is $k+1$-fold repetition code and thus a k-deletion correcting code that protects $H(R'(\mathbf{c}))$, the hash function $H(R'(\mathbf{c}))$ can be recovered from $(d_{n+N_0+N_1+1}, \ldots, d_{n+N_0+N_1+N_2-k})$.

Similarly, $(d_{n+N_0+1}, \ldots, d_{n+N_0+N_1-k})$ is a length $N_1 - k$ subsequence of $(\mathcal{E}(\mathbf{c})_{n+N_0+1}, \ldots, \mathcal{E}(\mathbf{c})_{n+N_0+N_1}) = R'(\mathbf{c})$. From Lemma 8.2.3, the function $R'(\mathbf{c})$ can be recovered from $H(R'(\mathbf{c}))$ and $(d_{n+N_0+1}, \ldots, d_{n+N_0+N_1-k})$. Hence Statement 1 holds.

We now prove Statement 2. Note that $R'(\mathbf{c})$ contains hashes $(f(\mathbb{1}_{sync}(T(\mathbf{c}))) \bmod p(T(\mathbf{c})), p(T(\mathbf{c})))$ and $Hash_k(T(\mathbf{c}))$. Moreover, $(d_1, \ldots, d_{n+N_0-k})$ is a length $n + N_0 - k$ subsequence of $(\mathcal{E}(\mathbf{c})_1, \ldots, \mathcal{E}(\mathbf{c})_{n+N_0}) = T(\mathbf{c})$. According to Lemma 3.2.2, the synchronization vector $\mathbb{1}_{sync}(T(\mathbf{c}))$ can be recovered from hash
$(f(\mathbb{1}_{sync}(T(\mathbf{c}))) \bmod p(T(\mathbf{c})), p(T(\mathbf{c})))$ and $(d_1, \ldots, d_{n+N_0-k})$, by exhaustively searching over all length $n + N_0$ supersequence $\mathbf{c}'$ of $(d_1, \ldots, d_{n+N_0-k})$ such that $f(\mathbb{1}_{sync}(\mathbf{c}')) = f(\mathbb{1}_{sync}(T(\mathbf{c})))$. Then we have that $\mathbb{1}_{sync}(T(\mathbf{c})) = \mathbb{1} + sync(\mathbf{c}')$. Since by Lemma 3.2.4, $T(\mathbf{c})$ is a k-dense sequence, by Lemma 3.2.3, it can be recovered from $\mathbb{1}_{sync}(T(\mathbf{c}))$, $Hash_k(T(\mathbf{c}))$, and the length $n + N_0 - k$ subsequence $(d_1, \ldots, d_{n+N_0-k})$ of $T(\mathbf{c})$. Finally, the sequence $\mathbf{c}$ can be recovered from $T(\mathbf{c})$ by Lemma 3.2.4. Hence Statement 2 holds and $\mathbf{c}$ can be recovered.

The encoding complexity of $\mathcal{E}(\mathbf{c})$ is $O(n^{2k+1})$, which comes from brute force search for integer $p(T(\mathbf{c}))$. The decoding complexity is $O(n^k + 1)$, which comes from brute force search for the correct $\mathbb{1}_{sync}(T(\mathbf{c}))$, given $f(\mathbb{1}_{sync}(T(\mathbf{c}))) \bmod p(T(\mathbf{c}))$ and $p(T(\mathbf{c}))$.

3.4 Protecting the Synchronization Vectors

In this section we present a hash function with size $4k \log n + o(\log n)$ to protect the synchronization vector $\mathbb{1}_{sync}(\mathbf{c})$ from k deletions in $\mathbf{c}$ and prove Lemma 3.2.2. We first prove Lemma 3.2.1, which is decomposed to Proposition 3.4.1 and Proposition 3.4.2. In Proposition 3.4.1 we present an upper bound on the radius of the deletion ball for the synchronization vector. In Proposition 3.4.2, we prove that the higher order parity check helps correct multiple deletions for sequences in which the there is a 0-run of length at least $3k - 1$ between any two 1's. Since $\mathbb{1}_{sync}(\mathbf{c})$ is such a sequence, we conclude that the higher order parity check helps recover $\mathbb{1}_{sync}(\mathbf{c})$.

After obtaining a bound on the difference between the higher order parity checks of two ambiguous sequence, we then apply Proposition 3.4.2 on the synchronization vector $\mathbb{1}_{sync}(\mathbf{c})$ to prove Lemma 3.2.1, which replaces the higher order parity checks in Proposition 3.4.2 by the higher parity checks modulo a numbers. After proving Lemma 3.2.1, we use Lemma 3.2.7 to further compress the size of the higher order parity check that protects $\mathbb{1}_{sync}(\mathbf{c})$ and then prove Lemma 3.2.2.

Proposition 3.4.1. *For* $\mathbf{c}, \mathbf{c}' \in \{0,1\}^n$, *if* $\mathbf{c}' \in \mathcal{B}_k(\mathbf{c})$, *then* $\mathbb{1}_{sync}(\mathbf{c}') \in \mathcal{B}_{3k}(\mathbb{1}_{sync}(\mathbf{c}))$.

Proof. Since $\mathbf{c}' \in \mathcal{B}_k(\mathbf{c})$, the sequences $\mathbf{c}'$ and $\mathbf{c}$ share a common subsequence after k deletions in both. We now show that a single deletion in $\mathbf{c}$ causes at most two deletions and one insertion in its synchronization vector $\mathbb{1}_{sync}(\mathbf{c})$. We first show that a deletion in $\mathbf{c}$ can destroy and generate at most 1 synchronization pattern. This is because for any synchronization pattern that is destroyed or generated, there must be a deletion that occurs within the synchronization pattern. Hence any two destroyed or generated synchronization patterns cannot be caused by the same deletion. Therefore, we need to consider four cases in total. Let $\mathbf{d}'$ be the subsequence of $\mathbf{c}$ after a single deletion.

1. The deletion destroys a synchronization pattern $(c_{i+1}, \ldots, c_{i+3k+\lceil \log k \rceil+4})$ for some i and no synchronization pattern is generated. Then the sequence $\mathbb{1}_{sync}(\mathbf{d}')$ can be obtained by deleting the 1 entry $\mathbb{1}_{sync}(\mathbf{c})_{i+3k}$ in $\mathbb{1}_{sync}(\mathbf{c})$.

2. The deletion generates a new synchronization pattern $(c'_{i'+1}, \ldots, c'_{i'+3k+\lceil \log k \rceil+4})$ for some i' and destroys a synchronization pattern $(c_{i+1}, \ldots, c_{i+3k+\lceil \log k \rceil+4})$. The sequence $\mathbb{1}_{sync}(\mathbf{d}')$ can be obtained by deleting the 1 entry $\mathbb{1}_{sync}(\mathbf{c})_{i+3k}$ and the 0 entry $\mathbb{1}_{sync}(\mathbf{c})_{i+3k-1}$ in $\mathbb{1}_{sync}(\mathbf{c})$ and inserting a 1 entry at $\mathbb{1}_{sync}(\mathbf{c})_{i'+3k}$.

3. The deletion generates a new synchronization pattern $(c'_{i'+1}, \ldots, c'_{i'+3k+\lceil \log k \rceil+4})$ for some i' and no synchronization pattern is destroyed. Then the $\mathbb{1}_{sync}(\mathbf{d}')$ can be obtained by deleting two 0 entries $\mathbb{1}_{sync}(\mathbf{c})_{i'+3k}$ and $\mathbb{1}_{sync}(\mathbf{c})_{i'+3k+1}$ in $\mathbb{1}_{sync}(\mathbf{c})$ and inserting a 1 entry at $\mathbb{1}_{sync}(\mathbf{c})_{i'+3k}$.

4. No synchronization pattern is generated or destroyed. Then $\mathbb{1}_{sync}(\mathbf{d}')$ can be obtained by deleting a 0 entry $\mathbb{1}_{sync}(\mathbf{c})_j$, where j is the location of the deletion.

In summary, in each of the above cases, a single deletion in $\mathbf{c}$ causes at most two deletions and one insertion in $\mathbb{1}_{sync}(\mathbf{c})$. Hence k deletions in $\mathbf{c}$ and $\mathbf{c}'$ cause at most $2k$ deletions and k insertions in $\mathbb{1}_{sync}(\mathbf{c})$ and $\mathbb{1}_{sync}(\mathbf{c}')$ respectively. According to Lemma 3.2.6, we have that $\mathbb{1}_{sync}(\mathbf{c}') \in \mathcal{B}_{3k}(\mathbb{1}_{sync}(\mathbf{c}))$ when $\mathbf{c}' \in \mathcal{B}_k(\mathbf{c})$. Hence, Proposition 3.4.1 is proved. □

Let $\mathcal{R}_m$ be the set of length n sequences in which there is a 0 run of length at least $m-1$ between any two 1's. Any two 1's in a sequence $\mathbf{c} \in \mathcal{R}_m$ have index distance at least m. The following lemma shows that the sequences in $\mathcal{R}_{3k}$ can be protected using higher order parity checks. Note that compared to the higher order parity checks $f(\mathbf{c})$, the higher order parity checks in the following proposition do not have modulo operations.

Proposition 3.4.2. *For sequences* $\mathbf{c}, \mathbf{c}' \in \mathcal{R}_{3k}$*, if* $\mathbf{c}' \in \mathcal{B}_{3k}(\mathbf{c})$ *and* $\mathbf{c} \cdot \mathbf{m}^{(e)} = \mathbf{c}' \cdot \mathbf{m}^{(e)}$ *for* $e \in [0, 6k]$*, then* $\mathbf{c} = \mathbf{c}'$.

Proof. We first compute the difference $\mathbf{c} \cdot \mathbf{m}^{(e)} - \mathbf{c}' \cdot \mathbf{m}^{(e)}$, $e \in [0, 6k]$. Since $\mathbf{c}' \in \mathcal{B}_{3k}(\mathbf{c})$, there exist two subsets $\boldsymbol{\delta} = \{\delta_1, \ldots, \delta_{3k}\} \subset [1, n]$ and $\boldsymbol{\delta}' = \{\delta'_1, \ldots, \delta'_{3k}\} \subset [1, n]$ such that deleting bits with indices $\boldsymbol{\delta}$ and $\boldsymbol{\delta}'$ respectively from $\mathbf{c}$ and $\mathbf{c}'$ results in the same length $n-3k$ subsequence, i.e., $(c_i : i \notin \boldsymbol{\delta}) = (c'_i : i \notin \boldsymbol{\delta}')$. Let $\boldsymbol{\Delta} = \{i : c_i = 1\}$ and $\boldsymbol{\Delta}' = \{i : c'_i = 1\}$ be the indices of 1 entries in $\mathbf{c}$ and $\mathbf{c}'$ respectively. Let $S_1 = \boldsymbol{\Delta} \cap \boldsymbol{\delta}$ be the indices of 1 entries that are deleted in $\mathbf{c}$. Then $S_1^c = \boldsymbol{\Delta} \cap ([1, n] \backslash \boldsymbol{\delta})$ denotes the indices of 1 entries that are not deleted. Similarly, let $S_2 = \boldsymbol{\Delta}' \cap \boldsymbol{\delta}'$ and $S_2^c = \boldsymbol{\Delta}' \cap ([1, n] \backslash \boldsymbol{\delta}')$ be the indices of 1 entries that are deleted and not in $\mathbf{c}'$ respectively. Let the elements in $\boldsymbol{\delta} \cup \boldsymbol{\delta}'$ be ordered by $1 \leq p_1 \leq p_2 \leq \ldots \leq p_{6k} \leq n$. Denote $p_0 = 0$ and $p_{6k+1} = n$. Then we have that

$$\begin{aligned}
&\mathbf{c} \cdot \mathbf{m}^{(e)} - \mathbf{c}' \cdot \mathbf{m}^{(e)} \\
=&\sum_{\ell \in \boldsymbol{\Delta}} \mathbf{m}^{(e)}_\ell - \sum_{\ell \in \boldsymbol{\Delta}'} \mathbf{m}^{(e)}_\ell \\
=&\sum_{\ell \in \boldsymbol{\Delta}} (\sum_{i=1}^{\ell} i^e) - \sum_{\ell \in \boldsymbol{\Delta}'} (\sum_{i=1}^{\ell} i^e) \\
=&\sum_{i=1}^{n} (\sum_{\ell \in \boldsymbol{\Delta} \cap [i,n]} i^e) - \sum_{i=1}^{n} (\sum_{\ell \in \boldsymbol{\Delta}' \cap [i,n]} i^e) \\
=&\sum_{i=1}^{n} (|\boldsymbol{\Delta} \cap [i,n]| - |\boldsymbol{\Delta}' \cap [i,n]|) i^e \\
=&\sum_{i=1}^{n} (|S_1 \cap [i,n]| + |S_1^c \cap [i,n]| - |S_2 \cap [i,n]| \\
&- |S_2^c \cap [i,n]|) i^e
\end{aligned}$$

$$
\begin{aligned}
=&\sum_{j=0}^{6k}\sum_{i=p_j+1}^{p_{j+1}}(|S_1\cap[i,n]|-|S_2\cap[i,n]|+|S_1^c\cap[i,n]|\\
&-|S_2^c\cap[i,n]|)i^e\\
\overset{(a)}{=}&\sum_{j=0}^{6k}\sum_{i=p_j+1}^{p_{j+1}}(|S_1\cap[p_{j+1},n]|\\
&-|S_2\cap[p_{j+1},n]|+|S_1^c\cap[i,n]|-|S_2^c\cap[i,n]|)i^e, \qquad (3.5)
\end{aligned}
$$

where (a) holds since by definition of p_j, there is no deleted 1 entry in interval $(p_j,p_{j+1})=\{p_j+1,\ldots,p_{j+1}-1\}$, $j\in[0,6k]$. In the following we show

Statement 1: $-1\le|S_1^c\cap[i,n]|-|S_2^c\cap[i,n]|\le 1$ for $i\in[1,n]$.

Statement 2: For each interval $(p_j,p_{j+1}]=\{p_j+1,\ldots,p_{j+1}\}, j=0,\ldots,6k$, we have either $|S_1^c\cap[i,n]|-|S_2^c\cap[i,n]|\le 0$ for all $i\in(p_j,p_{j+1}]$ or $|S_1^c\cap[i,n]|-|S_2^c\cap[i,n]|\ge 0$ for all $i\in(p_j,p_{j+1}]$.

We first prove **Statement 1**. Note that deleting bits with indices $\boldsymbol{\delta}$ in $\mathbf{c}$ and deleting bits with indices $\boldsymbol{\delta}'$ in $\mathbf{c}'$ result in the same subsequence. Hence, for every $i\in S_1^c$, there is a unique corresponding index $i'\in S_2^c$ such that the two 1 entries c_i and $c'_{i'}$ end in the same location after deletions, i.e., $i-|\boldsymbol{\delta}\cap[1,i-1]|=i'-|\boldsymbol{\delta}'\cap[1,i'-1]|$. This implies that $|i'-i|\le 3k$. Fix integers i and i'. Then by definition of i and i', for every $x\in S_1^c\cap[i+1,n]$, there is a unique corresponding $y\in S_2^c\cap[i'+1,n]$ such that the two 1 entries c_x and c'_y end in the same location after deletions. Therefore, we have that $|S_1^c\cap[i+1,n]|=|S_2^c\cap[i'+1,n]|$, and thus that $|S_1^c\cap[i,n]|=|S_2^c\cap[i',n]|$. If $i'\ge i$, then

$$
\begin{aligned}
&|S_1^c\cap[i,n]|-|S_2^c\cap[i,n]|\\
=&|S_1^c\cap[i,n]|-|S_2^c\cap[i',n]|-|S_2^c\cap[i,i'-1]|\\
=&-|S_2^c\cap[i,i'-1]|\\
\overset{(a)}{\ge}&-|S_2^c\cap[i,i+3k-1]|\\
\overset{(b)}{\ge}&-1,
\end{aligned}
$$

where (a) follows from the fact that $i'\le i+3k$ and (b) follows from the fact that $\mathbf{c},\mathbf{c}'\in\mathcal{R}_{3k}$. Also we have that $|S_1^c\cap[i,n]|-|S_2^c\cap[i,n]|=-|S_2^c\cap[i,i'-1]|\le 0$. Hence when $i'\le i$, we have that $-1\le|S_1^c\cap[i,n]|-|S_2^c\cap[i,n]|\le 0$. Similarly, when $i' < i$, we have that

$$\begin{aligned}
&|S_1^c \cap [i,n]| - |S_2^c \cap [i,n]| \\
=&|S_1^c \cap [i,n]| - |S_2^c \cap [i',n]| + |S_2^c \cap [i', i-1]| \\
=&|S_2^c \cap [i', i-1]| \\
\leq&|S_2^c \cap [i', i'+3k-1]| \\
\leq&1,
\end{aligned}$$

and that $|S_1^c \cap [i,n]| - |S_2^c \cap [i,n]| = |S_2^c \cap [i', i-1]| \geq 0$. Therefore, we have that $0 \leq |S_1^c \cap [i,n]| - |S_2^c \cap [i,n]| \leq 1$ when $i' < i$. Thus **Statement 1** is proved.

We now prove **Statement 2** by contradiction. Suppose on the contrary, there exist $i_1, i_2 \in (p_j, p_{j+1}]$ such that $i_1 < i_2$ and

$$\begin{aligned}
(|S_1^c \cap [i_1,n]| - |S_2^c \cap [i_1,n]|)(|S_1^c \cap [i_2,n]| - |S_2^c \cap [i_2,n]|) \\
< 0
\end{aligned}$$

From **Statement 1** we have that $|S_1^c \cap [i_1,n]| - |S_2^c \cap [i_1,n]| \in [-1,1]$ and that $|S_1^c \cap [i_2,n]| - |S_2^c \cap [i_2,n]| \in [-1,1]$. Hence by symmetry it can be assumed that $|S_1^c \cap [i_1,n]| - |S_2^c \cap [i_1,n]| = -1$ and $|S_1^c \cap [i_2,n]| - |S_2^c \cap [i_2,n]| = 1$. As shown in proof of **Statement 1**, for every element $i \in S_1^c$, there is a corresponding element $i' \in S_2^c$ such that the two 1 entries c_i and $c'_{i'}$ end in the same location after deletions. Hence, for $y = \min_{i \in S_2^c \cap [i_1,n]} i$, there exists an integer $x \in S_1^c$ such that the two 1 entries c_x and c'_y are in the same location after deletions, i.e., $x - |\boldsymbol{\delta} \cap [1, x-1]| = y - |\boldsymbol{\delta}' \cap [1, y-1]|$. Since $|S_1^c \cap [i_1,n]| - |S_2^c \cap [i_1,n]| = -1$, we have that $x \in S_1^c \cap [1, i_1 - 1]$. Otherwise, we have that $x \in S_1^c \cap [i_1, n]$ and for every integer $i' \in S_2^c \cap (y, n]$, there exists an integer $i \in S_1^c \cap (x, n]$ such that c_i and $c_{i'}$ end up in the same location after deletions. This implies that $|S_1^c \cap [i_1,n]| - |S_2^c \cap [i_1,n]| \geq 0$, contradicting the fact that $|S_1^c \cap [i_1,n]| - |S_2^c \cap [i_1,n]| = -1$. Therefore,

$$\begin{aligned}
i_1 - |\boldsymbol{\delta} \cap [1, i_1 - 1]| >& i_1 - 1 - |\boldsymbol{\delta} \cap [1, i_1 - 1]| \\
\geq& x - |\boldsymbol{\delta} \cap [1, x-1]| \\
=& y - |\boldsymbol{\delta}' \cap [1, y-1]| \\
\geq& i_1 - |\boldsymbol{\delta}' \cap [1, i_1 - 1]|,
\end{aligned}$$

which implies that

$$|\boldsymbol{\delta} \cap [1, i_1 - 1]| < |\boldsymbol{\delta}' \cap [1, i_1 - 1]|. \tag{3.6}$$

Similarly, from $|S_1^c \cap [i_2, n]| - |S_2^c \cap [i_2, n]| = 1$ we have that

$$|\boldsymbol{\delta} \cap [1, i_2 - 1]| > |\boldsymbol{\delta}' \cap [1, i_2 - 1]|. \tag{3.7}$$

Eq. (3.6) and Eq. (3.7) implies that

$$\begin{aligned} & |\boldsymbol{\delta} \cap [1, i_2 - 1]| - |\boldsymbol{\delta} \cap [1, i_1 - 1]| \\ \geq & |\boldsymbol{\delta}' \cap [1, i_2 - 1]| + 1 - |\boldsymbol{\delta}' \cap [1, i_1 - 1]| + 1 \\ \geq & 2. \end{aligned} \tag{3.8}$$

However, since $i_1, i_2 \in (p_j, p_{j+1}]$ and no deletion occurs in the interval $(p_j, p_{j+1}]$, we have that $|\boldsymbol{\delta} \cap [1, i_1]| = |\boldsymbol{\delta} \cap [1, i_2 - 1]|$ and $|\boldsymbol{\delta}' \cap [1, i_1]| = |\boldsymbol{\delta}' \cap [1, i_2 - 1]|$, which implies that

$$\begin{aligned} & |\boldsymbol{\delta} \cap [1, i_2 - 1]| - |\boldsymbol{\delta} \cap [1, i_1 - 1]| \\ \leq & |\boldsymbol{\delta} \cap [1, i_2 - 1]| - |\boldsymbol{\delta} \cap [1, i_1]| + 1 \\ = & 1, \end{aligned}$$

contradicting Eq. (3.8). Hence there do not exist different integers $i_1, i_2 \in (p_j, p_{j+1}]$ such that

$$\begin{aligned} (|S_1^c \cap [i_1, n]| - |S_2^c \cap [i_1, n]|)(|S_1^c \cap [i_2, n]| - |S_2^c \cap [i_2, n]|) \\ < 0. \end{aligned}$$

Hence **Statement 2** is proved.

Now we continue to prove Proposition 3.4.2. Denote

$$s_i \triangleq |S_1 \cap [i, n]| - |S_2 \cap [i, n]| + |S_1^c \cap [i, n]| - |S_2^c \cap [i, n]|. \tag{3.9}$$

Note that no deletion occurs in the interval $(p_j, p_{j+1}]$, it follows that

$$\begin{aligned} & |S_1 \cap [i, n]| - |S_2 \cap [i, n]| \\ = & |S_1 \cap [p_{j+1}, n]| - |S_2 \cap [p_{j+1}, n]| \end{aligned} \tag{3.10}$$

for $i \in (p_j, p_{j+1}]$. Combining (3.10) with **Statement 1** and **Statement 2**, we conclude that for each interval $(p_j, p_{j+1}]$, $j \in \{0, \ldots, 6k\}$, either $s_i \geq 0$ for all $i \in (p_j, p_{j+1}]$ or $s_i \leq 0$ for all $i \in (p_j, p_{j+1}]$. Let $\mathbf{x} = (x_0, \ldots, x_{6k}) \in \{-1, 1\}^{6k+1}$ be a vector defined by

$$x_i = \begin{cases} -1, & \text{if } s_j < 0 \text{ for some } j \in (p_i, p_{i+1}] \\ 1, & \text{else.} \end{cases}.$$

Then from Eq. (3.5) and Eq. (3.9), the difference $\mathbf{c} \cdot \mathbf{m}^{(e)} - \mathbf{c}' \cdot \mathbf{m}^{(e)}$ is given by

$$\mathbf{c} \cdot \mathbf{m}^{(e)} - \mathbf{c}' \cdot \mathbf{m}^{(e)} = \sum_{j=0}^{6k}(\sum_{i=p_j+1}^{p_{j+1}} |s_i| i^e) x_j. \tag{3.11}$$

Let A be a $6k+1 \times 6k+1$ matrix with entries defined by $A_{e,j} = \sum_{i=p_{j-1}+1}^{p_j} |s_i| i^{e-1}$ for $e, j \in [1, 6k+1]$. If $\mathbf{c} \cdot \mathbf{m}^{(e)} = \mathbf{c}' \cdot \mathbf{m}^{(e)}$ for $e \in [0, 6k]$, we have the following linear equation

$$\begin{aligned} A\boldsymbol{x} =& \begin{bmatrix} \sum_{i=p_0+1}^{p_1} |s_i| i^0 & \dots & \sum_{i=p_{6k}+1}^{p_{6k+1}} |s_i| i^0 \\ \vdots & \ddots & \vdots \\ \sum_{i=p_0+1}^{p_1} |s_i| i^{6k} & \dots & \sum_{i=p_{6k}+1}^{p_{6k+1}} |s_i| i^{6k} \end{bmatrix} \begin{bmatrix} x_0 \\ \vdots \\ x_{6k} \end{bmatrix} \\ =&0, \end{aligned} \tag{3.12}$$

with a solution $x_i \in \{-1, 1\}$ for $i \in [0, 6k]$. We show that this is impossible unless A is a zero matrix. Suppose on the contrary that A is nonzero, let $j_1 < \ldots < j_Q$ be the indices of all nonzero columns of A. Let A^* be a submatrix of A, obtained by choosing the intersection of the first Q rows and columns with indices $j_1, \ldots, j_Q$. Then taking the first Q linear equations from the equation set (3.13) and noting that the nonzero columns in A are the $j_1, \ldots, j_Q$-th columns, we have that

$$\begin{aligned} &A^*\boldsymbol{x}' \\ =& \begin{bmatrix} \sum_{i=p_{j_1-1}+1}^{p_{j_1}} |s_i| i^0 & \dots & \sum_{i=p_{j_{Q-1}}+1}^{p_{j_Q}} |s_i| i^0 \\ \vdots & \ddots & \vdots \\ \sum_{i=p_{j_1-1}+1}^{p_{j_1}} |s_i| i^{Q-1} & \dots & \sum_{i=p_{j_{Q-1}}+1}^{p_{j_Q}} |s_i| i^{Q-1} \end{bmatrix} \begin{bmatrix} x_{j_1} \\ \vdots \\ x_{j_Q} \end{bmatrix} \\ =&0. \end{aligned} \tag{3.13}$$

The determinant of A^* is given by

$$\begin{aligned} &\det(A^*) \\ =&\det \begin{pmatrix} \sum_{i=p_{j_1-1}+1}^{p_{j_1}} |s_i| i^0 & \dots & \sum_{i=p_{j_{Q-1}}+1}^{p_{j_Q}} |s_i| i^0 \\ \vdots & \ddots & \vdots \\ \sum_{i=p_{j_1-1}+1}^{p_{j_1}} |s_i| i^{Q-1} & \dots & \sum_{i=p_{j_{Q-1}}+1}^{p_{j_Q}} |s_i| i^{Q-1} \end{pmatrix} \\ \overset{(a)}{=} & \sum_{\substack{i_1 \in (p_{j_1-1}, p_{j_1}], \ldots, \\ i_Q \in (p_{j_Q-1}, p_{j_Q}]}} \det \begin{pmatrix} |s_{i_1}| i_1^0 & \dots & |s_{i_Q}| i_Q^0 \\ \vdots & \ddots & \vdots \\ |s_{i_1}| i_1^{Q-1} & \dots & |s_{i_Q}| i_Q^{Q-1} \end{pmatrix} \end{aligned}$$

$$\overset{(b)}{=} \sum_{\substack{i_1 \in (p_{j_1-1}, p_{j_1}], \dots, \\ i_Q \in (p_{j_{Q-1}}, p_{j_Q}]}} \left[\prod_{q=1}^{Q} |s_{i_q}| \det \begin{pmatrix} i_1^0 & \dots & i_Q^0 \\ \vdots & \ddots & \vdots \\ i_1^{Q-1} & \dots & i_Q^{Q-1} \end{pmatrix} \right]$$

$$\overset{(c)}{=} \sum_{\substack{i_1 \in (p_{j_1-1}, p_{j_1}], \dots, \\ i_Q \in (p_{j_{Q-1}}, p_{j_Q}]}} [\prod_{q=1}^{Q} |s_{i_q}| \prod_{1 \le m < \ell \le Q} (i_\ell - i_m)], \tag{3.14}$$

where equality (a) follows from the multi-linearity of the determinant

$$\begin{aligned} &\det([\mathbf{v}_1 \ \dots \ a\mathbf{v}_i + b\mathbf{v} \ \dots \ \mathbf{v}_q]) \\ =&a \det([\mathbf{v}_1 \ \dots \ \mathbf{v}_i \ \dots \ \mathbf{v}_q]) \\ &+ b \det([\mathbf{v}_1 \ \dots \ \mathbf{v}_{i-1} \ \mathbf{v} \ \mathbf{v}_{i+1} \ \dots \ \mathbf{v}_q]) \end{aligned}$$

for any integers q and i and q-dimensional vectors $\mathbf{v}_1, \dots, \mathbf{v}_q$, $\mathbf{v}$. Equality (b) follows from the linearity of the determinant

$$\det([\mathbf{v}_1 \ \dots \ a\mathbf{v}_i \ \dots \ \mathbf{v}_q]) = a \det([\mathbf{v}_1 \ \dots \ \mathbf{v}_i \ \dots \ \mathbf{v}_q])$$

for any integers q and i and q-dimensional vectors $\mathbf{v}_1, \dots, \mathbf{v}_q$. Equality (c) follows from the determinant of Vandermonde matrix

$$\det \begin{pmatrix} i_1^0 & \dots & i_q^0 \\ \vdots & \ddots & \vdots \\ i_1^{q-1} & \dots & i_q^{q-1} \end{pmatrix} = \prod_{1 \le m < \ell \le q} (i_\ell - i_m)$$

for any integers q, $i_1, \dots, i_q$. . The determinant $det(A^*)$ is positive since $i_\ell > i_m$ for $\ell > m$. and for $i_1 \in (p_{j_1-1}, p_{j_1}], \dots, i_Q \in (p_{j_{Q-1}}, p_{j_Q}]$. Note that all the columns of A^* are nonzero. Therefore, the linear equation $A^*\boldsymbol{x}' = 0$ does not have nonzero solutions, contradicting the fact that $\boldsymbol{x}' = (x_{j_1}, \dots, x_{j_Q}) \in \{-1, 1\}^Q$. Hence A is a zero matrix, meaning that

$$\begin{aligned} &|S_1 \cap [i,n]| - |S_2 \cap [i,n]| + |S_1^c \cap [i,n]| - |S_2^c \cap [i,n]| \\ =&|\mathbf{\Delta} \cap [i,n]| - |\mathbf{\Delta}' \cap [i,n]| = 0 \end{aligned}$$

for $i \in \{1, \dots, n\}$. This implies $\mathbf{\Delta} = \mathbf{\Delta}'$ and thus $\mathbf{c} = \mathbf{c}'$. Hence Proposition 3.4.2 is proved. □

Proof of Lemma 3.2.1

We are now ready to prove Lemma 3.2.1, which states that $\mathbb{1}_{sync}(\mathbf{c}) = \mathbb{1}_{sync}(\mathbf{c}')$ for sequences $\mathbf{c}$ and $\mathbf{c}' \in \mathcal{B}_k(\mathbf{c})$ satisfying $f(\mathbb{1}_{sync}(\mathbf{c})) = f(\mathbb{1}_{sync}(\mathbf{c}))$. From

Proposition 3.4.1 we have that $\mathbb{1}_{sync}(\mathbf{c}') \in \mathcal{B}_{3k}(\mathbb{1}_{sync}(\mathbf{c}))$. Then, it is not hard to see that $(\mathbb{1}_{sync}(\mathbf{c}')_i, \ldots, \mathbb{1}_{sync}(\mathbf{c}')_n) \in \mathcal{B}_{3k}((\mathbb{1}_{sync}(\mathbf{c})_i, \ldots, \mathbb{1}_{sync}(\mathbf{c})_n))$. This implies that $||\mathbf{\Delta} \cap [i,n]| - |\mathbf{\Delta}' \cap [i,n]|| \leq 3k$, where $\mathbf{\Delta} = \{i : \mathbb{1}_{sync}(\mathbf{c})_i = 1\}$ and $\mathbf{\Delta}' = \{i : \mathbb{1}_{sync}(\mathbf{c}')_i = 1\}$. According to the forth line in Eq. (3.5), we have that

$$\begin{aligned}
&|\mathbb{1}_{sync}(\mathbf{c}) \cdot \mathbf{m}^{(e)} - \mathbb{1}_{sync}(\mathbf{c}') \cdot \mathbf{m}^{(e)}| \\
=&|\sum_{i=1}^{n}(|\mathbf{\Delta} \cap [i,n]| - |\mathbf{\Delta}' \cap [i,n]|)i^e|, \\
\leq&\sum_{i=1}^{n} 3ki^e \\
<&3kn^{e+1}.
\end{aligned} \tag{3.15}$$

If $f(\mathbb{1}_{sync}(\mathbf{c})) = f(\mathbb{1}_{sync}(\mathbf{c}'))$, then

$$\mathbb{1}_{sync}(\mathbf{c}) \cdot \mathbf{m}^{(e)} \equiv \mathbb{1}_{sync}(\mathbf{c}') \cdot \mathbf{m}^{(e)} \bmod 3kn^{e+1} \tag{3.16}$$

for $e \in [0, 6k]$. Equations (3.16) and (3.15) imply that $\mathbb{1}_{sync}(\mathbf{c}) \cdot \mathbf{m}^{(e)} = \mathbb{1}_{sync}(\mathbf{c}') \cdot \mathbf{m}^{(e)}$ for $e \in [0, 6k]$. Since $\mathbb{1}_{sync}(\mathbf{c}') \in \mathcal{B}_{3k}(\mathbb{1}_{sync}(\mathbf{c}))$ and $\mathbb{1}_{sync}(\mathbf{c}), \mathbb{1}_{sync}(\mathbf{c}') \in \mathcal{R}_{3k}$, from Proposition 3.4.2 we conclude that $\mathbb{1}_{sync}(\mathbf{c}) = \mathbb{1}_{sync}(\mathbf{c}')$. Hence Lemma 3.2.1 is proved.

Proof of Lemma 3.2.2

Based on Lemma 3.2.1, we now show Lemma 3.2.2. Specifically, we show that there exists a function $p : \{0,1\}^n \to [1, 2^{2k\log n + o(\log n)}]$ such that $\mathbb{1}_{sync}(\mathbf{c}) = \mathbb{1}_{sync}(\mathbf{c}')$ for sequences $\mathbf{c}$ and $\mathbf{c}' \in \mathcal{B}_k(\mathbf{c})$ satisfying $(f(\mathbb{1}_{sync}(\mathbf{c})) \bmod p(\mathbf{c}), p(\mathbf{c})) = (f(\mathbb{1}_{sync}(\mathbf{c}')) \bmod p(\mathbf{c}'), p(\mathbf{c}'))$.

Lemma 3.2.1 implies that $f(\mathbb{1}_{sync}(\mathbf{c})) \neq f(\mathbb{1}_{sync}(\mathbf{c}'))$ for $\mathbf{c}' \in \mathcal{B}_k(\mathbf{c}) \backslash \{\mathbf{c}\}$, if $\mathbb{1}_{sync}(\mathbf{c}) \neq \mathbb{1}_{sync}(\mathbf{c}')$. Hence $|f(\mathbb{1}_{sync}(\mathbf{c})) - f(\mathbb{1}_{sync}(\mathbf{c}'))| \neq 0$ for $\mathbf{c}' \in \mathcal{B}_k(\mathbf{c}) \backslash \{\mathbf{c}\}$, where $f(\mathbb{1}_{sync}(\mathbf{c}))$ and $f(\mathbb{1}_s ync(\mathbf{c}'))$ denote the integer presentation of their vector form. The integers are in the range $[0, (3k)^{6k+1} n^{(3k+1)(6k+1)} - 1]$. According to Lemma 3.2.7, the number of divisors of $|f(\mathbb{1}_{sync}(\mathbf{c})) - f(\mathbb{1}_{sync}(\mathbf{c}'))|$ is upper bounded by

$$\begin{aligned}
&2^{2[(3k+1)(6k+1)\ln n + (6k+1)\ln 3k]/\ln((3k+1)(6k+1)\ln n + (6k+1)\ln 3k)} \\
&= 2^{o(\log n)},
\end{aligned}$$

where the equality holds since $k = o(\sqrt{\log\log n})$. For any sequence $\mathbf{c} \in \{0,1\}^n$, let

$$\begin{aligned}\mathcal{P}(\mathbf{c}) = \{&p : p \text{ divides } |f(\mathbb{1}_{sync}(\mathbf{c}')) - f(\mathbb{1}_{sync}(\mathbf{c}))| \\ &\text{for some } \mathbf{c}' \in \mathcal{B}_k(\mathbf{c})\backslash\{\mathbf{c}\} \text{ such that } \mathbb{1}_{sync}(\mathbf{c}) \neq \mathbb{1}_{sync}(\mathbf{c}')\}\end{aligned}$$

be the set of all divisors of the numbers $\{|f(\mathbb{1}_{sync}(\mathbf{c}')) - f(\mathbb{1}_{sync}(\mathbf{c}))| : \mathbf{c}' \in \mathcal{B}_k(\mathbf{c})\backslash\{\mathbf{c}\}$ and $\mathbb{1}_{sync}(\mathbf{c}) \neq \mathbb{1}_{sync}(\mathbf{c}')\}$. Since $|\mathcal{B}_k(\mathbf{c})| \leq \binom{n}{k}^2 2^k \leq 2n^{2k}$, we have that

$$\begin{aligned}|\mathcal{P}(\mathbf{c})| \leq& 2n^{2k}2^{o(\log n)} \\ =& 2^{2k\log n + o(\log n)}.\end{aligned}$$

Therefore, there exists a number $p(\mathbf{c}) \in [1, 2^{2k\log n+o(\log n)}]$ such that $p(\mathbf{c})$ does not divide $|f(\mathbb{1}_{sync}(\mathbf{c}')) - f(\mathbb{1}_{sync}(\mathbf{c}))|$ for all $\mathbf{c}' \in \mathcal{B}_k(\mathbf{c})\backslash\{\mathbf{c}\}$ satisfying $\mathbb{1}_{sync}(\mathbf{c}) \neq \mathbb{1}_{sync}(\mathbf{c}')$. Hence, if $f(\mathbb{1}_{sync}(\mathbf{c}')) \equiv f(\mathbb{1}_{sync}(\mathbf{c})) \bmod p(\mathbf{c})$ and $\mathbf{c}' \in \mathcal{B}_k(\mathbf{c})$, we have that $p(\mathbf{c})$ divides $|f(\mathbb{1}_{sync}(\mathbf{c}')) - f(\mathbb{1}_{sync}(\mathbf{c}))|$, and thus that $\mathbb{1}_{sync}(\mathbf{c}') = \mathbb{1}_{sync}(\mathbf{c})$. This completes the proof of Lemma 3.2.2.

3.5 Hash for k-Dense Sequences

In this section, we present a hash function of size $4k\log n + o(\log n)$ bits for correcting k deletions in a k-dense sequence $\mathbf{c}$, when the synchronization vector $\mathbb{1}_{sync}(\mathbf{c})$ is known. This proves Lemma 3.2.3. Recall that a sequence $\mathbf{c}$ is k-dense if there is a 0 run of length at most L between any two 1's in the synchronization vector $\mathbb{1}_{sync}(\mathbf{c})$, where L is given in (3.4).

Let the indices of the 1 entries in $\mathbb{1}_{sync}(\mathbf{c})$ be $t_1 < t_2 < \ldots < t_J$, where $J = \sum_{i=1}^{n} \mathbb{1}_{sync}(\mathbf{c})_i$ is the number of 1 entries in $\mathbb{1}_{sync}(\mathbf{c})$. For notation convenience, let $t_0 = 0$ and $t_{J+1} = n+1$. Split $\mathbf{c}$ into blocks $\mathbf{a}_0, \ldots, \mathbf{a}_J$, where

$$\mathbf{a}_j = (c_{t_j+1}, c_{t_j+2}, \ldots, c_{t_{j+1}-1}) \tag{3.17}$$

for $j \in [0, J]$. The blocks are separated by the synchronization patterns. Since $\mathbf{c}$ is k-dense, the length $|\mathbf{a}_j|$ of $\mathbf{a}_j$ is at most L. The goal is to protect all blocks $\mathbf{a}_j$, $j \in [0, J]$ and then $\mathbf{c}$.

We will show that given $\mathbb{1}_{sync}(\mathbf{c})$ and a length $n-k$ subsequence $\mathbf{d}$ of $\mathbf{c}$, most of the blocks $\mathbf{a}_j$ can be recovered with up to $2k$ block errors. This is done by noticing that no deletion occurs in most of the blocks and their boundary, which are marked by the synchronization patterns in $\mathbf{c}$. These blocks with no deletions inside are not destructed and appear in $\mathbf{d}$ with bits indices decreased by an integer at most k. They can be identified by looking at the synchronization patterns in $\mathbf{d}$.

For blocks that are not recovered, we show that they can be recovered with up to k deletion errors. Then we use the k-deletion correcting hash function $H(\mathbf{a}_j)$, $j \in [0, J]$ defined in Lemma 8.2.3 to correct the block errors. The size of the hash $H(\mathbf{a}_j)$ is at most $L/\lceil \log n \rceil (2k \log\log n + O(1))$ bits, since the length of $\mathbf{a}_j$ is at most L. Note that the recovered blocks $\mathbf{a}_j$ result in the right hash $H(\mathbf{a}_j)$. It suffices to protect the hashes $H(\mathbf{a}_j)$, $j \in [0, J]$ using a Reed-Solomon code. Define the hash function $Hash_k$ as follows.

$$Hash_k(\mathbf{c}) = RS_{2k}((H(\mathbf{a}_0), \dots, H(\mathbf{a}_J))), \tag{3.18}$$

where $RS_{2k}(\mathbf{c})$ is the redundancy of a systematic Reed-Solomon code (see e.g., [79] for an introduction to the Reed-Solomon code) protecting the length $J + 1$ sequence $(H(\mathbf{a}_0), \dots, H(\mathbf{a}_J))$ from $2k$ symbol substitution errors. Note that the redundancy of a k-error correcting Reed-Solomon code of length n and alphabet size $q \geq n - 1$ is $2k \log q$ bits [79]. The symbols $H(\mathbf{a}_j)$,$j \in [0, J]$ have alphabet size at most $2^{\lceil (L/\lceil \log n \rceil) \rceil (2k \log\log n + O(1))}$ and can be represented using $\lceil (L/\lceil \log n \rceil) \rceil (2k \log\log n + O(1))$ bits. The length of $Hash_k(\mathbf{c})$ is

$$\max\{4k \log(J + 1), 4k\lceil (L/\lceil \log n \rceil) \rceil (2k \log\log n + O(1))\},$$

which equals $4k \log n + o(\log n)$ when $k = o(\sqrt{\log\log n})$.

We now present the following procedure that recovers $\mathbf{c}$ from its length $n - k$ subsequence $\mathbf{d}$, given the hash function $Hash_k(\mathbf{c})$ and the synchronization vector $\mathbb{1}_{sync}(\mathbf{c})$.

1. **Step 1:** Let $\mathbb{1}_{sync}(\mathbf{d}) \in \{0, 1\}^{n-k}$ be the synchronization vector of $\mathbf{d}$. The indices of 1 entries in $\mathbb{1}_{sync}(\mathbf{c})$ are known and given by $1 \leq t_1 < \dots < t_J \leq n$. Let $t_0 = 0$ and $t_{J+1} = n + 1$.

2. **Step 2:** Let $\mathbb{1}_{sync}(\mathbf{d})_0 = \mathbb{1}_{sync}(\mathbf{d})_{n+1-k} = 1$. For each $j \in [0, J]$, if there exist two numbers $t'_j \in [t_j - k, t_j]$ and $t'_{j+1} \in [t_{j+1} - k, t_{j+1}]$ such that $\mathbb{1}_{sync}(\mathbf{d})_{t'_j} = \mathbb{1}_{sync}(\mathbf{d})_{t'_{j+1}} = 1$ and $t'_{j+1} - t'_j = t_{j+1} - t_j$, let $\mathbf{a}'_j = (d_{t'_j+1}, d_{t'_j+2}, \dots, d_{t'_{j+1}-1})$. Else let $\mathbf{a}'_j = 0$.

3. **Step 3:** Apply the Reed-Solomon decoder to recover $H(\mathbf{a}_j)$ from $(H(\mathbf{a}'_0), \dots, H(\mathbf{a}'_J), Hash_k(\mathbf{c}))$, where $\mathbf{a}_j$ is defined in (3.19), $j \in [0, J]$.

4. **Step 4:** Let $\mathbf{b}_j = (d_{t_j+1}, d_{t_j+2}, \ldots, d_{t_{j+1}-k-1})$ and recover $\mathbf{a}_j$ by using $\mathbf{b}_j$ and $H(\mathbf{a}_j)$. Then

$$\mathbf{c} = (\mathbf{a}_1, 1, \mathbf{a}_2, 1, \ldots, \mathbf{a}_{J-1}, 1, \mathbf{a}_J).$$

The following example illustrates how to recover the blocks with at most $2k$ block errors.

Example 3.5.1. *Let $k = 2$, $n = 6$, and sequence $\mathbf{c}$ the same as in Example 3.1.1, i.e.,*

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

The indices of the 1 entries in $\mathbb{1}_{sync}(\mathbf{c})$ are $\{6, 14, 23, 30\}$. Then the tuple $(t_0, t_1, t_2, t_3, t_4, t_5) = (0, 6, 14, 23, 30, 36)$ is known at the decoder. Suppose the 2 deletions occurs at the first and the last bits, resulting in a subsequence

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

Then the synchronization vector of $\mathbf{d}$ is given by

$$\mathbb{1}_{sync}(\mathbf{d}) = (0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,$$
$$0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),$$

with indices of the 1 entries given by $\{0, 6, 13, 22, 34\}$, where it is assumed in the decoding procedure that $\mathbb{1}_{sync}(\mathbf{d})_0 = \mathbb{1}_{sync}(\mathbf{d})_{34} = 1$. Then we have that $t'_0 = 0$, $t'_1 = 6$, $t'_2 = 13$, and $t'_3 = 22$, which implies that the intervals $[t_0, t_1]$ and $[t_2, t_3]$ are good. Hence, we can recover 2 blocks $\mathbf{a}'_0 = (1, 1, 1, 1, 1)$ and $\mathbf{a}'_2 = (1, 1, 1, 1, 1, 0, 1, 0)$. The number of block errors is $3 \leq 2k$.

We now prove that the decoding procedure obtains the correct $\mathbf{c}$. Since $\mathbf{c}_{t_j} = \mathbb{1}_{sync}(\mathbf{c})_{t_j} = 1$ for $j \in [1, J]$, it suffices to show that $\mathbf{a}_j$, $j \in [0, J]$ can be recovered correctly. Note that $(d_{t_j+1}, \ldots, d_{t_{j+1}-k-1})$ is a length $|\mathbf{a}_j| - k$ subsequence of $(c_{t_j+1}, \ldots, c_{t_{j+1}-1}) = \mathbf{a}_j$. Hence $\mathbf{a}_j$ can be correctly decoded given $\mathbf{b}_j$ and $H(\mathbf{a}_j)$, $j \in [0.J]$. It is then left to recover $H(\mathbf{a}_j)$ for $j \in [0, J]$. To this end, we show that there are at most $2k$ indices j, such that $\mathbf{a}'_j \neq \mathbf{a}_j$. Since $H(\mathbf{a}_j)$ can be computed for correct blocks $\mathbf{a}_j$, there are at most $2k$ symbol errors in the sequence $(H(\mathbf{a}_0), \ldots, H(\mathbf{a}_J))$, which can be corrected given the Reed-Solomon code redundancy $Hash_k(\mathbf{c})$.

Let t''_j, $j \in [1, J]$ be the index of $\mathbf{c}_{t_j}$ in $\mathbf{d}$ after deletions in $\mathbf{c}$, where $t''_j = -1$ if $\mathbf{c}_{t_j}$ is deleted. Let $t''_0 = 0$ and $t''_{J+1} = n + 1 - k$. The interval $[t_j, t_{j+1}]$, $j \in [0.J]$ is called *good* if $\mathbb{1}_{sync}(\mathbf{d})_{t''_j} = \mathbb{1}_{sync}(\mathbf{d})_{t''_{j+1}} = 1$ and $t''_{j+1} - t''_j = t_{j+1} - t_j$. We now show that $\mathbf{a}'_j = \mathbf{a}_j$ if the interval $[t_j, t_{j+1}]$ is *good*. Note that the bits $d_{t''_j}$ and $d_{t''_{j+1}}$ come from c_{t_j} and $c_{t_{j+1}}$ respectively after deletions in $\mathbf{c}$. Hence if $[t_j, t_{j+1}]$ is good, we have that $t''_{j+1} - t''_j = t_{j+1} - t_j$, and thus that

$$(d_{t''_j+1}, \dots, d_{t''_{j+1}-1}) = (c_{t_j+1}, \dots, c_{t_{j+1}-1}) = \mathbf{a}_j. \tag{3.19}$$

Moreover, we have that $\mathbb{1}_{sync(\mathbf{d})_{t''_j}} = \mathbb{1}_{sync}(\mathbf{d})_{t''_{j+1}} = 1$. Since $t_j - t''_j \leq k$ and $t_{j+1} - t''_{j+1} \leq k$ by definition of t''_j and t''_{j+1}, it follows from Step 2 in the decoding procedure that $t'_j = t''_j, t'_{j+1} = t''_{j+1}$, and $\mathbf{a}'_j = (d_{t''_j+1}, \dots, d_{t''_{j+1}-1})$. Hence from (3.19), we conclude that $\mathbf{a}'_j = \mathbf{a}_j$ when the interval $[t_j, t_{j+1}]$ is good.

Next, we show that a deletion can destroy at most 2 *good* intervals. Notice that if no deletion occurs in $\mathbf{c}$, then all intervals $[t_j, t_{j+1}]$, $j \in [0, J]$ are good. If no deletions occur in the interval $[t_{j-1}, t_{j+2}]$, then $\mathbb{1}_{sync}(\mathbf{d})_{t''_j} = \mathbb{1}_{sync}(\mathbf{d})_{t''_{j+1}} = 1$ and $t''_{j+1} - t''_j = t_{j+1} - t_j$, where t''_j is the index of c_{t_j} in $\mathbf{d}$ after deletions. Hence the interval $[t_j, t_{j+1}]$ is good when no deletion occurs in $[t_{j-1}, t_{j+2}]$. It follows that a deletion that occurs in interval $[t_j, t_{j+1}]$ can destroy at most 3 good intervals $[t_{j-1}, t_j]$, $[t_j, t_{j+1}]$, and $[t_{j+1}, t_{j+2}]$. We prove that the deletion in $[t_j, t_{j+1}]$ can destroy at most two of them. If the deletion in $[t_j, t_{j+1}]$ does not destroy the synchronization pattern $(c_{t_j-3k+1}, \dots, c_{t_j+\lceil \log k \rceil+4})$, then it destroys at most two good intervals $[t_j, t_{j+1}]$ and $[t_{j+1}, t_{j+2}]$. If the deletion in $[t_j, t_{j+1}]$ destroys the synchronization pattern $(c_{t_j-3k+1}, \dots, c_{t_j+\lceil \log k \rceil+4})$, then it destroys the pattern $(c_{t_j}, \dots, c_{t_j+\lceil \log k \rceil+4}) = \mathbf{1}^{\lceil \log k \rceil+5}$ and the synchronization pattern $(c_{t_{j+1}-3k+1}, \dots, c_{t_{j+1}+\lceil \log k \rceil+4})$ is not affected. As a result, the good interval $[t_{j+1}, t_{j+2}]$ is not destroyed by the deletion in $[t_j, t_{j+1}]$. At most two good intervals $[t_{j-1}, t_j]$ and $[t_j, t_{j+1}]$ are destroyed. Therefore, a deletion affects at most two *good* intervals. This implies that k deletions result in at most $2k$ block errors $\mathbf{a}'_j \neq \mathbf{a}_j$. Therefore, the sequence $\mathbf{c}$ can be recovered.

3.6 Generating *k-Dense* Sequences

In this section we present an algorithm to compute the map $T(\mathbf{c})$, which transforms any sequence $\mathbf{c} \in \{0, 1\}^n$ into a k-*dense* sequence, and thus proves Lemma 3.2.4. Let $\mathbf{1}^x$ and $\mathbf{0}^y$ denote sequences of consecutive x 1's and consecutive y 0's, respectively. We first show in Proposition 3.6.1 that any sequence $\mathbf{c}$ satisfying the following two properties is a k-*dense* sequence. Then, the algorithm for computing computing $T(\mathbf{c})$ can be decomposed into two parts. In the first part, we generate a sequence $T_1(\mathbf{c})$ that satisfies Property 1. In the second part, we use $T_1(\mathbf{c})$ to compute $T(\mathbf{c})$ that satisfies both properties.

Property 1. *Every length* $B \triangleq (\lceil \log k \rceil + 5)2^{\lceil \log k \rceil + 9} \lceil \log n \rceil$ *interval of* $\mathbf{c}$ *contains the pattern* $\mathbf{1}^{\lceil \log k \rceil + 5}$*, i.e., for any integer* $i \in [1, n - B + 1]$*, there exists an integer* $j \in [i, i+B-\lceil \log k \rceil - 5]$ *such that* $(c_j, c_{j+1}, \ldots, c_{j+\lceil \log k \rceil + 4}) = \mathbf{1}^{\lceil \log k \rceil + 5}$.

Property2. *Every length* $R \triangleq (3k + \lceil \log k \rceil + 4)(\lceil \log n \rceil + 9 + \lceil \log k \rceil)$ *interval of* $\mathbf{c}$ *contains a length* $3k + \lceil \log k \rceil + 4$ *subinterval that does not contain the pattern* $\mathbf{1}^{\lceil \log k \rceil + 5}$*, i.e., for any integer* $i \in [1, n - R + 1]$*, there exists an integer* $j \in [i, i+R-3k-\lceil \log k \rceil - 4]$*, such that* $(c_m, c_{m+1}, \ldots, c_{m+\lceil \log k \rceil + 4}) \neq \mathbf{1}^{\lceil \log k \rceil + 5}$ *for every* $m \in [j, j + 3k - 1]$.

Proposition 3.6.1. *If a sequence* $\mathbf{c}$ *satisfies Property 1 and Property 2, then it is a* k-dense *sequence.*

Proof. Let the locations of the 1 entries in $\mathbb{1}_{sync}(\mathbf{c})$ be $t_1 < \ldots < t_J$. Let $t_0 = 0$ and $t_{J+1} = n+1$. From Definition 3.2.1, it suffices to show that $t_{i+1} - t_i \leq B + R + 1 = L + 1$ for any $i \in [0, J]$.

According to Property 2, there exists an index $j^* \in [t_i, t_i + R - 3k - \lceil \log k \rceil - 4]$, such that $(c_m, c_{m+1}, \ldots, c_{m+\lceil \log k \rceil + 4}) \neq \mathbf{1}^{\lceil \log k \rceil + 5}$ for every $m \in [j^*, j^* + 3k - 1]$. According to Property 1, there exists an integer $x \in [j^* + 1, j^* + B]$ such that $(c_x, c_{x+1}, \ldots, c_{x+\lceil \log k \rceil + 4}) = \mathbf{1}^{\lceil \log k \rceil + 5}$. Let $\ell = \min\{x \geq j^* : (c_x, c_{x+1}, \ldots, c_{x+\lceil \log k \rceil + 4}) = \mathbf{1}^{\lceil \log k \rceil + 5}\}$. Then we have that $\ell \neq j^*$, $\ell \leq x \leq j^* + B$, and thus that $\ell \in [j^*+1, j^*+B]$. In addition, $(c_m, c_{m+1}, \ldots, c_{m+\lceil \log k \rceil + 4}) \neq \mathbf{1}^{\lceil \log k \rceil + 5}$ for every $m \in [j^*, \ell) = \{j^*, \ldots, \ell - 1\}$. By definition of j^*, we have that $\ell - j^* \geq 3k$. Since $(c_\ell, c_{\ell+1}, \ldots, c_{\ell+\lceil \log k \rceil + 4}) = \mathbf{1}^{\lceil \log k \rceil + 5}$, we have that $\mathbb{1}_{sync}(\mathbf{c})_\ell = 1$. Therefore, we conclude that

$$\begin{aligned} t_{i+1} - t_i &\leq \ell - t_i \\ &\leq j^* + B - t_i \\ &\leq R + B + 1 \\ &= L + 1. \end{aligned}$$

□

Generating Sequences Satisfying Property 1

Given a sequence $\mathbf{c} \in \{0,1\}^n$, we now generate $T_1(\mathbf{c}) \in \{0,1\}^{n+2\lceil \log k \rceil+10}$ that satisfies Property 1. The idea is to repeatedly delete the length B subsequences of $\mathbf{c}$ that do not contain the pattern $\mathbf{1}^{\lceil \log k \rceil+5}$, and append length B subsequences containing $\mathbf{1}^{\lceil \log k \rceil+5}$ to the end, without losing the information of the deleted subsequences. The deleting and appending procedure repeats until no length B subsequence with no $\mathbf{1}^{\lceil \log k \rceil+5}$ pattern is found. Notice that any binary sequence containing no pattern $\mathbf{1}^{\lceil \log k \rceil+5}$ can be regarded as a sequence of symbols with alphabet size $2^{\lceil \log k \rceil+5} - 1$. Hence, such binary sequence can be compressed to a shorter binary sequence. In this way, we can add the index of the deleted subsequence and the pattern $\mathbf{1}^{\lceil \log k \rceil+5}$ to the compressed sequence. The sequence keeps the same size after the deleting and appending procedure.

Note that the above procedure keeps appending length B subsequences to the end. Hence the suffix of $T_1(\mathbf{c})$ are appended bits. To guarantee that these appended bits are not deleted in the procedure, we keep track of the end index of the non-appended bits n' and always delete the bits with indices at most n'. To deal with cases when a length B subsequence to be deleted overlaps with the appended bits, we delete only non-appended bits from it. Then, we append shorter subsequences such that the total length does not change. The decoder detects the shorter appended subsequences by looking at the length of the 1 run in it.

Before presenting the details of encoding and decoding, we need the following proposition, which states that the length B binary sequences containing no $\mathbf{1}^{\lceil \log k \rceil+5}$ pattern can be compressed to shorter binary sequences.

Proposition 3.6.2. *Let $\mathcal{S}$ be the set of sequences $\mathbf{b} \in \{0,1\}^B$ such that $(b_i, \ldots, b_{i+\lceil \log k \rceil+4}) \neq \mathbf{1}^{\lceil \log k \rceil+5}$ for every $i \in [1, B - \lceil \log k \rceil - 4]$. There exists an invertible map $\phi : \mathcal{S} \rightarrow \{0,1\}^{B-\lceil \log n \rceil - 2\lceil \log k \rceil - 12}$, such that both ϕ and its inverse ϕ^{-1} can be computed in $O(B)$ time.*

Proof. For any $\mathbf{b} \in \mathcal{S}$, the procedure for computing $\phi(\mathbf{b})$ is as follows. Split $\mathbf{b}$ into $2^{\lceil \log k \rceil+9}\lceil \log n \rceil$ blocks of length $(\lceil \log k \rceil + 5)$. Since each block is not $\mathbf{1}^{\lceil \log k \rceil+5}$, it can be represented by a symbol of alphabet size $2^{\lceil \log k \rceil+5} - 1$. Therefore, the

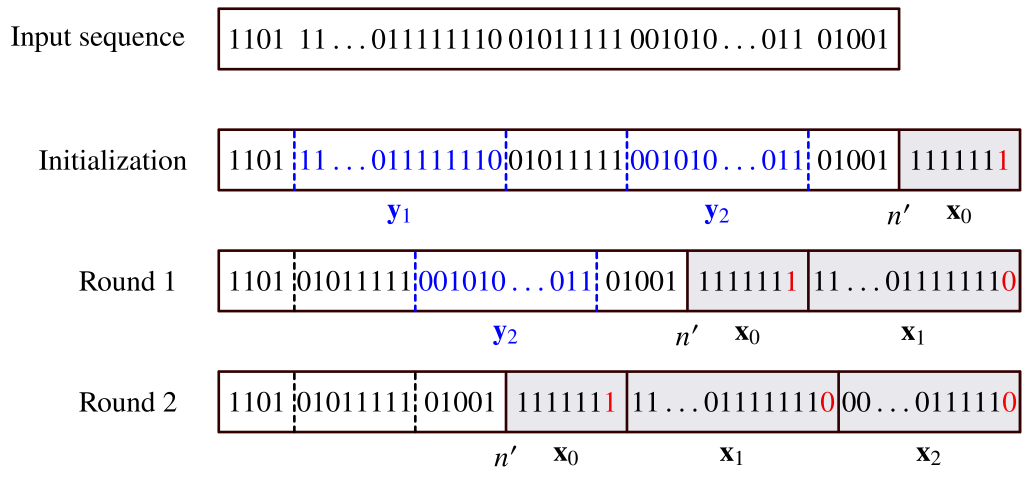

Figure 3.3: An example of how the encoding in Proposition 3.6.3 proceeds.

sequence $\mathbf{b}$ can be uniquely represented by a sequence $\mathbf{v}$ of $2^{\lceil \log k\rceil+9}\lceil \log n\rceil$ symbols, each having alphabet size $2^{\lceil \log k\rceil+5}-1$. Convert $\mathbf{v}$ into a binary sequence $\phi(\mathbf{b})$. Then $\phi(\mathbf{b})$ can be represented by a binary sequence with length

$$
\begin{aligned}
&\lceil \log_2 [(2^{\lceil \log k\rceil+5}-1)^{2^{\lceil \log k\rceil+9}\lceil \log n\rceil}]\rceil\\
=&\lceil \log_2 [(1-1/2^{\lceil \log k\rceil+5})^{2^{\lceil \log k\rceil+9}\lceil \log n\rceil}]\rceil\\
&+(\lceil \log k\rceil+5)2^{\lceil \log k\rceil+9}\lceil \log n\rceil\\
\leq&16\lceil \log n\rceil \log_2 [(1-1/2^{\lceil \log k\rceil+5})^{2^{\lceil \log k\rceil+5}}]+B+1\\
\overset{(a)}{\leq}&-16\lceil \log n\rceil \log_2 e+B+1\\
\leq&B-16\lceil \log n\rceil+1\\
\leq&B-\lceil \log n\rceil-2\lceil \log k\rceil-12,
\end{aligned}
$$

where (a) follows from the fact that the function $(1-1/x)^x$ is increasing in x for $x>1$ and that $\lim_{x\rightarrow\infty}(1-1/x)^x=1/e$. Therefore, $\phi(\mathbf{b})$ can be represented by $B-\lceil \log n\rceil-2\lceil \log k\rceil-12$ bits. The inverse map ϕ^{-1} can be computed by converting $\phi(\mathbf{b})$ back to a length $2^{\lceil \log k\rceil+9}\lceil \log n\rceil$ sequence $\mathbf{v}$ of alphabet size $2^{\lceil \log k\rceil+5}-1$. Then, concatenate the binary representation of symbols in $\mathbf{v}$, we obtain $\mathbf{b}$.

The complexity for computing ϕ or ϕ^{-1} is that of converting binary sequences to sequences of alphabet size $2^{\lceil \log k\rceil+5}-1$ or vice versa, which is $O(B)$. □

With the function ϕ defined in Proposition 3.6.2, the following proposition provides details of the encoding/decoding for computing $T_1(\mathbf{c})$. An example illustrating the encoding in Proposition 3.6.3 is presented in Fig. 3.3.

Proposition 3.6.3. *For integers k and $n > k$, there exists an invertible map $T_1 : \{0,1\}^n \to \{0,1\}^{n+2\lceil \log k \rceil+10}$, computable in $O(n^2 k \log n \log^2 k)$ time, such that $T_1(\mathbf{c})$ satisfies Property 1. Moreover, either*

$$(T_1(\mathbf{c})_{n+\lceil \log k \rceil+6}, \ldots, T_1(\mathbf{c})_{n+2\lceil \log k \rceil+10}) = \mathbf{1}^{\lceil \log k \rceil+5}$$

or

$$(T_1(\mathbf{c})_{n+\lceil \log k \rceil+5}, \ldots, T_1(\mathbf{c})_{n+2\lceil \log k \rceil+9}) = \mathbf{1}^{\lceil \log k \rceil+5}.$$

Proof. For a sequence $\mathbf{c} \in \{0,1\}^n$, the encoding procedure for computing $T_1(\mathbf{c})$ is as follows.

1. **Initialization:** Let $T_1(\mathbf{c}) = \mathbf{c}$. Append $\mathbf{1}^{2\lceil \log k \rceil+10}$ to the end of the sequence $T_1(\mathbf{c})$. Let $n' = n$. Go to Step 1.

2. **Step 1:** If there exists an integer $i \in [1, n']$ such that

$$(T_1(\mathbf{c})_j, T_1(\mathbf{c})_{j+1}, \ldots, T_1(\mathbf{c})_{j+\lceil \log k \rceil+4}) \neq \mathbf{1}^{\lceil \log k \rceil+5}$$

   for every $j \in [i, i+B-\lceil \log k \rceil - 5]$, go to Step 2. Else go to Step 4.

3. **Step 2:** If $i > n' - B + 1$, go to Step 3. Else, delete $(T_1(\mathbf{c})_i, \ldots, T_1(\mathbf{c})_{i+B-1})$ from $T_1(\mathbf{c})$ and append

$$(i, \phi(T_1(\mathbf{c})_i, \ldots, T_1(\mathbf{c})_{i+B-1}), 0, \mathbf{1}^{2\lceil \log k \rceil+10}, 0)$$

   to the end of $T_1(\mathbf{c})$, where the appended i is encoded by $\lceil \log n \rceil$ binary bits. Let $n' = n' - B$. Go to Step 1.

4. **Step 3:** Delete $(T_1(\mathbf{c})_i, \ldots, T_1(\mathbf{c})_{n'})$ from $T_1(\mathbf{c})$ and append

$$(i, \phi(T_1(\mathbf{c})_i, \ldots, T_1(\mathbf{c})_{n'}, \mathbf{0}^{i+B-n'-1}), 0, \mathbf{1}^{2\lceil \log k \rceil+10-(i+B-n'-1)}, 0)$$

   to the end of $T_1(\mathbf{c})$. Let $n' = i - 1$, where the appended i is encoded by $\lceil \log n \rceil$ binary bits. Go to Step 1.

5. **Step 4:** Output $T_1(\mathbf{c})$.

In the encoding procedure, Step 2 and Step 3 are the deleting and appending operation described above, while Step 3 deals with the case when the deleted subsequence overlaps with the appended bits. Note that the index of the deleted subsequence is provided in the appended subsequence, so that the decoder can recover the deleted subsequence, given the appended subsequence. According to Proposition 3.6.2 and the fact that the index i has size $\lceil \log n \rceil$ bits, the lengths of the deleted and appended subsequences in Step 2 or Step 3 are equal. Hence, the sequence $T_1(\mathbf{c})$ keeps constant and is $n + 2\lceil \log k \rceil + 10$.

We first show that the integer n' is the split index of appended bits and non-appended bits. Specifically, $(T_1(\mathbf{c})_{n'+1}, \ldots, T_1(\mathbf{c})_{n+2\lceil \log k \rceil+10})$ are the appended bits and $(T_1(\mathbf{c})_1, \ldots, T_1(\mathbf{c})_{n'})$ are non-appended bits that remain after deleting operations in Step 2 and Step 3. In addition, $(T_1(\mathbf{c})_{n'+1}, \ldots, T_1(\mathbf{c})_{n'+2\lceil \log k \rceil+10}) = \mathbf{1}^{2\lceil \log k \rceil+10}$ are the bits appended in the Initialization step. The claim holds in the Initialization step. Note that in each round of Step 2 or Step 3, the deleted bits have indices at most n' and the integer n' decreases by the length of deleted subsequence. Hence, the claim always holds.

We now show that the output sequence $T_1(\mathbf{c})$ satisfies Property 1.

We have shown that $(T_1(\mathbf{c})_{n'+1}, \ldots, T_1(\mathbf{c})_{n'+\lceil \log k \rceil+5}) = \mathbf{1}^{\lceil \log k \rceil+5}$. Then according to the if conditions in Step 1 and Step 2 that lead to Step 3, the integer i in Step 3 satisfies $1 \leq i + B - n' - 1 \leq \lceil \log k \rceil + 4$. Otherwise the subsequence $(T_1(\mathbf{c})_i, \ldots, T_1(\mathbf{c})_{i+B-1})$ contains $(T_1(\mathbf{c})_{n'+1}, \ldots, T_1(\mathbf{c})_{n'+\lceil \log k \rceil+5}) = \mathbf{1}^{\lceil \log k \rceil+5}$, which does not satisfy the if condition in Step 1. Therefore, the 1 run $\mathbf{1}^{2\lceil \log k \rceil+10-(i+B-n'-1)}$ in the appended subsequence in Step 3 has length at least $\lceil \log k \rceil + 6$. Thus, the $\mathbf{1}^{\lceil \log k \rceil+5}$ pattern appears in the subsequence appended in each round of Step 2 or Step 3. Moreover, the index distance between the two $\mathbf{1}^{\lceil \log k \rceil+5}$ patterns in two consecutively appended subsequences is at most $B - \lceil \log k \rceil - 5$. We conclude that for $i' > n'$, any length B subsequence $(T_1(\mathbf{c})_i, \ldots, T_1(\mathbf{c})_{i+B-1})$ contains the $\mathbf{1}^{\lceil \log k \rceil+5}$ pattern. Furthermore, note that for any $i \in [1, n']$, there exists some $j \in [i, i + B - \lceil \log k \rceil - 5]$ such that $T_1(\mathbf{c})_j = T_1(\mathbf{c})_{j+1} = \ldots = T_1(\mathbf{c})_{j+\lceil \log k \rceil+4} = 1$. Otherwise $T_1(\mathbf{c})_i$ is deleted in Step 2 or Step 3. Hence, the sequence $T_1(\mathbf{c})$ satisfies Property 1.

Note that the integer n' decreases in each round. Hence, the algorithm terminates within $O(n)$ rounds of Step 1, Step 2, and Step 3. Therefore, the search for the i satisfying the if condition in Step 1 takes $O(nB \log k)$ time. In addition, the deleting and appending operation in Step 2 or Step 3 takes at most $O(n + B)$ time. Hence, the total complexity is $O(n^2 k \log n \log^2 k)$.

Next, we show that either $(T_1(\mathbf{c})_{n+\lceil \log k\rceil+6}, \ldots, T_1(\mathbf{c})_{n+2\lceil \log k\rceil+10}) = \mathbf{1}^{\lceil \log k\rceil+5}$ or $(T_1(\mathbf{c})_{n+\lceil \log k\rceil+5}, \ldots, T_1(\mathbf{c})_{n+2\lceil \log k\rceil+9}) = \mathbf{1}^{\lceil \log k\rceil+5}$. The former holds when the appending operation only occurs in the Initialization step. Note that all subsequence appended in Step 2 or Step 3 ends with a 1 run with length at least $\lceil \log k\rceil + 6$ followed by a 0 bit. Hence the latter holds when the appending operation in Step 2 or Step 3 occurs.

The decoding follows a reverse procedure of the encoding, by repeatedly removing the appended subsequences and use them to recover the deleted subsequences. Since the appended subsequences contain the ϕ function and position of the deleted subsequences, the deleted subsequences can be recovered. The decoding stops when the end of the appended sequence becomes a 1 bit. The decoder determines the length of the appended subsequence by looking at the 1 run before the ending 0 bit. The decoding procedure that recovers $\mathbf{c}$ from $T_1(\mathbf{c})$ is given as follows.

1. **Initialization:** Let $\mathbf{c} = T_1(\mathbf{c})$ and go to Step 1.
2. **Step 1:** If $c_{n+2\lceil \log k\rceil+10} = 0$, find the length ℓ of the 1 run that ends with $c_{n+2\lceil \log k\rceil+9}$. Let i be the integer representation of $(c_{n+4\lceil \log k\rceil+21-B-\ell}, c_{n+4\lceil \log k\rceil+22-B-\ell}, \ldots, c_{n+4\lceil \log k\rceil+20-B-\ell+\lceil \log n\rceil})$. Let $\mathbf{b}$ be the sequence obtained by computing $\phi^{-1}(c_{n+4\lceil \log k\rceil+21-B-\ell+\lceil \log n\rceil}, c_{n+4\lceil \log k\rceil+22-B-\ell+\lceil \log n\rceil}, \ldots, c_{n+2\lceil \log k\rceil+8-\ell})$, where the function ϕ is defined in Proposition 3.6.2, Delete $(c_{n+4\lceil \log k\rceil+21-B-\ell}, c_{n+4\lceil \log k\rceil+22-B-\ell}, \ldots, c_{n+2\lceil \log k\rceil+10})$ from $\mathbf{c}$ and insert $(\mathbf{b}_1, \ldots, \mathbf{b}_{B-2\lceil \log k\rceil-10+\ell})$ at location i of $\mathbf{c}$. Repeat. Else delete $c_{n+1}, \ldots, c_{n+2\lceil \log k\rceil+10}$ and go to Step 2.
3. **Step 2:** Output $\mathbf{c}$.

In each round of Step 1, the decoder processes an appended subsequence of length B or less. It can be seen that the appended subsequences in the encoding procedure end with a 0 bit except for the one appended in the Initialization step. Hence if the end of an appended sequence is a 1 bit, the subsequence is the $\mathbf{1}^{2\lceil \log k\rceil+10}$ appended in the Initialization step of the encoding procedure. Moreover, since $\mathbf{1}^{2\lceil \log k\rceil+10}$ is the appended subsequence, the decoding procedure ends up in $\mathbf{1}^{2\lceil \log k\rceil+10}$ after processing all other appended subsequences.

Note that the encoding procedure consists of a series of deleting and appending operations. The decoding procedure exactly reverses the series of operations in the encoding procedure. Let $T_{1,i}(\mathbf{c})$, $i \in [0, I]$ be the sequence $T_1(\mathbf{c})$ obtained after the i-th deleting and appending operation in the encoding procedure, where I is the number of deleting and appending operations in total in the encoding procedure. We have that $T_{1,0}(\mathbf{c}) = \mathbf{c}$ and that $T_{1,I}(\mathbf{c})$ is the final output $T_1(\mathbf{c})$. Then the decoding procedure obtains $T_{1,I-i}(\mathbf{c})$, $i \in [0, I]$ after the i-th deleting and inserting operation. Hence we get the output $T_{1,I-I}(\mathbf{c}) = \mathbf{c}$ in the decoding procedure. □

Proof of Lemma 3.2.4

After having the sequence $T_1(\mathbf{c}) \in \{0,1\}^{n+2\lceil \log k\rceil+10}$ satisfying Property 1, we now use $T_1(\mathbf{c})$ to generate a sequence $T(\mathbf{c}) \in \{0,1\}^{n+3k+3\lceil \log k\rceil+15}$ that satisfies both Property 1 and Property 2. According to Proposition 3.6.1, $T(\mathbf{c})$ is a k-dense sequence and Lemma 3.2.4 is proved. The encoding of $T(\mathbf{c})$ follows a similar repetitive deleting and appending procedure to the encoding in Proposition 3.6.3. To satisfy Property 2, we repeatedly look for length R subsequences, every length $3k + \lceil \log k\rceil + 4$ subsequence of which contains $\mathbf{1}^{\lceil \log k\rceil+5}$ patterns, delete most part of it, and then append a subsequence that contains a length $3k + \lceil \log k\rceil + 4$ subsequence containing no $\mathbf{1}^{\lceil \log k\rceil+5}$ pattern. The appended subsequence has all information about the deleted sequence, including the index and the bits, and has the same length as that of the deleted subsequence. To this end, we need to construct a map that compresses a length $3k + \lceil \log k\rceil + 4$ sequence containing $\mathbf{1}^{\lceil \log k\rceil+5}$ patterns to a shorter sequence. Moreover, the compressed sequence does not contain $\mathbf{1}^{\lceil \log k\rceil+5}$ patterns. Then we can compress the deleted subsequence and add indices and markers as we did in Proposition 3.6.3. The end of an appended subsequence is marked by a 0 bit. We also keep track of the integer n', which is the end of non-appended bits, to guarantee that the appended bits are not deleted.

Note that we do not delete the whole length R subsequences, every length $3k + \lceil \log k\rceil + 4$ subsequence of which contains $\mathbf{1}^{\lceil \log k\rceil+5}$ patterns. Instead, we split the length R subsequence into blocks of length $3k+\lceil \log k\rceil+4$, each containing $\mathbf{1}^{\lceil \log k\rceil+5}$ patterns. Then the first and the last blocks remain and the blocks in the middle are deleted. This is to keep the sequence $T(\mathbf{c})$ satisfying Property 1 and protect the appended bits from being deleted.

The following proposition presents the compression map described above, which encodes a sequence containing $\mathbf{1}^{\lceil \log k\rceil+5}$ patterns into a shorter sequence without the $\mathbf{1}^{\lceil \log k\rceil+5}$ pattern. Similar to the encoding procedures that compute $T(\mathbf{c})$ and $T_1(\mathbf{c})$, the algorithm for computing the compression map follows a delete and append process, which repeatedly deletes $\mathbf{1}^{\lceil \log k \rceil+5}$ patterns and appends their indices and 0 runs to the end.

Proposition 3.6.4. *For an integer k, let $\mathbf{c} \in \{0,1\}^{3k+\lceil \log k \rceil+4}$ be a sequence such that $c_i = c_{i+1} = \ldots = c_{i+\lceil \log k \rceil+4} = 1$ for some $i \in [1,3k]$. There exists an invertible mapping $T_2 : \{0,1\}^{3k+\lceil \log k \rceil+4} \to \{0,1\}^{3k+\lceil \log k \rceil+3}$, such that $T_2(\mathbf{c})$ contains no $\lceil \log k \rceil + 5$ consecutive 1 bits. Both T_2 and its inverse T_2^{-1} are computable in $O(k^2 \log k)$ time.*

Proof. Given $\mathbf{c} \in \{0,1\}^{3k+\lceil \log k \rceil+4}$, the encoding procedure for computing $T_2(\mathbf{c})$ is as follows.

1. **Initialization:** Let $T_2(\mathbf{c}) = \mathbf{c}$. Append 0 to the end of the sequence $T_2(\mathbf{c})$. Find the smallest $i \in [1,3k]$ such that $T_2(\mathbf{c})_i = T_2(\mathbf{c})_{i+1} = \ldots = T_2(\mathbf{c})_{i+\lceil \log k \rceil+4} = 1$. Delete $(T_2(\mathbf{c})_i, \ldots, T_2(\mathbf{c})_{i+\lceil \log k \rceil+4})$ from $T_2(\mathbf{c})$ and append $(i, \mathbf{0}^{\lceil \log k \rceil+3-\lceil \log(3k) \rceil})$ to the end of $T_2(\mathbf{c})$, where the appended i is encoded by $\lceil \log 3k \rceil$ binary bits. Let $n' = 3k - 1$ and $i = 1$. Go to Step 1.
2. **Step 1:** If there exists an integer $i \leq n'$ such that $T_2(\mathbf{c})_i = T_2(\mathbf{c})_{i+1} = \ldots = T_2(\mathbf{c})_{i+\lceil \log k \rceil+4} = 1$, delete $(T_2(\mathbf{c})_i, \ldots, T_2(\mathbf{c})_{i+\lceil \log k \rceil+4})$ from $T_2(\mathbf{c})$ and append $(i, \mathbf{0}^{\lceil \log k \rceil+4-\lceil \log(3k) \rceil}, 1)$ to the end of $T_2(\mathbf{c})$. Let $n' = n' - \lceil \log k \rceil - 5$ and $i = 1$. Repeat. Else go to Step 2.
3. **Step 2:** Output $T_2(\mathbf{c})$.

There are deleting and appending operations in both the Initialization step and Step 1. In the Initialization step, the length of the deleted subsequence is larger than the length of the appended subsequence by 2. Hence after appending the 0 bit in the beginning, the length of $T_2(\mathbf{c})$ decreases by 1 after the Initialization step. The lengths of the deleted and appended subsequence in Step 1 are equal. Hence, the length of the sequence $T_2(\mathbf{c})$ keeps constant after the Initialization step and is

$$\begin{aligned} &3k + \lceil \log k \rceil + 4 + 1 - \lceil \log k \rceil - 5 + \lceil \log k \rceil + 3 \\ =&3k + \lceil \log k \rceil + 3. \end{aligned}$$

We show that $(T_2(\mathbf{c})_{n'+1}, \ldots, T_2(\mathbf{c})_{3k+\lceil \log k \rceil+3})$ are appended bits and $(T_2(\mathbf{c})_1, \ldots, T_2(\mathbf{c})_{n'})$ are non-appended bits remain in $T_2(\mathbf{c})$ after deletions. In particular, $T_2(\mathbf{c})_{n'+1} = 0$ is the bit appended in the beginning of the Initialization step. The claims hold after the Initialization step. Note that when $T_2(\mathbf{c})_{n'} = 0$, the integer i satisfying the if condition in Step i is at most $n' - \lceil \log k \rceil - 4$. Otherwise $(T_2(\mathbf{c})_i, \ldots, T_2(\mathbf{c})_{i+\lceil \log k \rceil+4}) \neq \mathbf{1}$. Therefore, the deleted bits in Step 1 have indices at most n'. Moreover, the integer n' decreases by the length of the deleted bits. Therefore, we conclude that $(T_2(\mathbf{c})_{n'+1}, \ldots, T_2(\mathbf{c})_{3k+\lceil \log k \rceil+3})$ consists of appended bits and that $T_2(\mathbf{c})_{n'+1} = 0$ is the bit appended at the beginning of the Initialization step. The appended bits $(T_2(\mathbf{c})_{n'+1}, \ldots, T_2(\mathbf{c})_{3k+\lceil \log k \rceil+3})$ are not deleted in the procedure.

We now show that $T_2(\mathbf{c})$ contains no $\mathbf{1}^{\lceil \log k \rceil+5}$ patterns. Note that the $\mathbf{1}^{\lceil \log k \rceil+5}$ patterns with indices at most n' are deleted in the encoding procedure. Since $T_2(\mathbf{c})_{n'+1} = 0$, there is no $\mathbf{1}^{\lceil \log k \rceil+5}$ pattern containing the bit $T_2(\mathbf{c})_{n'+1}$. Hence a $\mathbf{1}^{\lceil \log k \rceil+5}$ pattern has indices at least $n' + 1$. Moreover, since the $\lceil \log k \rceil + 4$-th bit in each appended subsequence is a 0 bit, there is no $\mathbf{1}^{\lceil \log k \rceil+5}$ pattern in the appended bits. Hence, no $\mathbf{1}^{\lceil \log k \rceil+5}$ with indices at least $n' + 1$ exists. This implies that $T_2(\mathbf{c})$ does not contain $\mathbf{1}^{\lceil \log k \rceil+5}$ patterns.

Since n' decreases in each step, the algorithm terminates within $O(k)$ iterations of Step 1. Since it takes $O(k \log k)$ to look for the integer i, the total complexity is $O(k^2 \log k)$.

The decoding $T_2^{-1}(\mathbf{c})$, which recovers $\mathbf{c}$ from $T_2(\mathbf{c})$, follows a reverse procedure of the encoding and is presented in the following.

1. **Initialization:** Let $\mathbf{c} = T_2(\mathbf{c})$ and go to Step 1.

2. **Step 1:** If $c_{3k+\lceil \log k \rceil+3} = 1$, let i be the integer representation of $(c_{3k-1}, c_{3k}, \ldots, c_{3k+\lceil \log 3k \rceil-2})$. Delete $(c_{3k-1}, c_{3k}, \ldots, c_{3k+\lceil \log k \rceil+3})$ from $\mathbf{c}$ and insert $\mathbf{1}^{\lceil \log k \rceil+5}$ at location i of $\mathbf{c}$. Repeat. Else go to Step 2.

3. **Step 2:** Let i be the decimal representation of $(c_{3k+1}, c_{3k+2}, \ldots, c_{3k+\lceil \log(3k) \rceil})$. Delete $(c_{3k}, c_{3k+2}, \ldots, c_{3k+\lceil \log k \rceil+3})$ from $\mathbf{c}$ and insert $\mathbf{1}^{\lceil \log k \rceil+5}$ at location i of $\mathbf{c}$. Output $\mathbf{c}$.

Note that in the encoding procedure, the appended subsequence in the Initialization Step ends with a 0. The appended subsequence in Step 1 ends with a 1. Hence the algorithm stops when $\mathbf{c}_{3k+\lceil \log k \rceil+3} = 0$ and all subsequences appended in Step 1 of the encoding have be processed.

Similar to the proof of correctness of decoding in Proposition 3.6.3, the decoding procedure exactly reverses the series of operations in the encoding procedure. Note that the appended subsequences contain the index of the deleted $\mathbf{1}^{\lceil \log k \rceil + 5}$ patterns. Let $T_{2,i}(\mathbf{c})$, $i \in [0, I]$ be the sequence obtained after the i-th deleting and appending operation in the encoding procedure, where I is the number of deleting and appending operations in total in the encoding procedure. Then $T_{2,i}(\mathbf{c})$ is the sequence obtained after the $I-i$-th deleting and inserting operation in the decoding procedure. Therefore, the decoding procedure recovers the sequence $\mathbf{c}$ after the I-th operation.

The complexity of the decoding has the same order $O(k^2 \log k)$ as that of the encoding. □

We are now ready to present the encoding and decoding procedures for computing $T(\mathbf{c})$, which generates k-dense sequences that satisfy Property 1 and Property 2. The encoding procedure is as follows.

1. **Initialization:** Let $T(\mathbf{c}) = T_1(\mathbf{c})$. Append $(\mathbf{0}^{3k}, \mathbf{1}^{\lceil \log k \rceil + 5})$ to the end of the sequence $T(\mathbf{c})$. Let $n' = n + 2\lceil \log k \rceil + 10$ (the length of $T_1(\mathbf{c})$). Go to Step 1.

2. **Step 1:** If there exists an integer $i \leq \min\{n', n+3k+3\lceil \log k \rceil+16-R\}$, such that for every $j \in [i, i + R - 3k - \lceil \log k \rceil - 4]$, there exists an integer $m \in [j, j + 3k - 1]$ satisfying $(T(\mathbf{c})_m, T(\mathbf{c})_{m+1}, \ldots, T(\mathbf{c})_{m+\lceil \log k \rceil + 4}) = \mathbf{1}^{\lceil \log k \rceil + 5}$, split $(T(\mathbf{c})_i, T(\mathbf{c})_{i+1}, \ldots, T(\mathbf{c})_{i+R-1})$ into $(\lceil \log n \rceil + 9 + \lceil \log k \rceil)$ blocks $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{\lceil \log n \rceil + 9 + \lceil \log k \rceil}$ of length $3k + \lceil \log k \rceil + 4$. Delete $(\mathbf{b}_2, \ldots, \mathbf{b}_{\lceil \log n \rceil + 8 + \lceil \log k \rceil})$ from $T(\mathbf{c})$ and append $(0, T_2(\mathbf{b}_2), T_2(\mathbf{b}_3), \ldots, T_2(\mathbf{b}_{\lceil \log n \rceil + 8 + \lceil \log k \rceil}), i + 3k + \lceil \log k \rceil + 4, \mathbf{1}^{\lceil \log k \rceil + 5}, 0)$ to the end of $T(\mathbf{c})$, where the appended $i + 3k + \lceil \log k \rceil + 4$ encoded by $\lceil \log n \rceil$ binary bits. Let $n' = n' - R + 6k + 2\lceil \log k \rceil + 8$. Repeat. Else go to Step 2.

3. **Step 2:** Output $T(\mathbf{c})$.

Note that the index $i+3k+\lceil \log k \rceil+4$ has size $\log n$ bits. Then from Proposition 3.6.4, it can be verified that the lengths of the deleted and appended subsequences in Step 1 are equal. Hence $T(\mathbf{c})$ keeps constant and is $n + 3k + 3\lceil \log k \rceil + 15$.

Similar to the encoding in Proposition 3.6.3 and Proposition 3.6.4, we show that the integer n' marks the end of the non-appended bits, i.e., the subsequence $(T(\mathbf{c})_{n'+1}, \ldots, (\mathbf{c})_{n+3k+3\lceil \log k \rceil + 15})$ consists of appended bits and $(T(\mathbf{c})_1, \ldots, T(\mathbf{c})_{n'})$ are non-appended bits that remain after deletions. In addition, we show that

$T(\mathbf{c})_{n'+1}, \ldots, T(\mathbf{c})_{n'+3k} = \mathbf{0}^{3k}$ and that the appended bits are not deleted. The claims hold in the Initialization step. Suppose the claims hold in the r-th round of Step 1. Then in the $r+1$-th round of Step 1, the integer i satisfying the if condition in Step 1 must be in the range $[1, n' - R + 3k + \lceil \log k \rceil + 4]$. Otherwise, since $T(\mathbf{c})_{n'+1}, \ldots, T(\mathbf{c})_{n'+3k} = \mathbf{0}^{3k}$ in the r-th round, we have an integer $n' \in [i, i + R - 3k - \lceil \log k \rceil - 4]$ such that $(T(\mathbf{c})_m, T(\mathbf{c})_{m+1}, \ldots, T(\mathbf{c})_{m+\lceil \log k \rceil + 4}) \neq \mathbf{1}^{\lceil \log k \rceil + 5}$ for every $m \in [n', n' + 3k - 1]$. This contradicts to the fact that i satisfies the if condition. Moreover, note that the block $\mathbf{b}_{\lceil \log n \rceil + 9 + \lceil \log k \rceil}$ in $(T(\mathbf{c})_i, \ldots, T(\mathbf{c})_{i+R-1})$ is not deleted in Step 1, which implies that the bits with indices at least $i + R - 3k - \lceil \log k \rceil - 4 \leq n'$ are not deleted. Note that n' decreases by the length of the deleted sequence in Step 1. We conclude that $(T(\mathbf{c})_{n'+1}, \ldots, (\mathbf{c})_{n+3k+3\lceil \log k \rceil + 15})$ consists of appended bits and these bits are not deleted. Specifically, $T(\mathbf{c})_{n'+1}, \ldots, T(\mathbf{c})_{n'+3k}$ are the bits appended in the Initialization step. By induction, the claims hold.

We now show by induction on the number of rounds r that $T(\mathbf{c})$ satisfies Property 1. From Proposition 3.6.3, the initial sequence $T(\mathbf{c}) = (T_1(\mathbf{c}), \mathbf{0}^{3k}, \mathbf{1}^{\lceil \log k \rceil + 5})$ satisfies Property 1. Hence the claim holds for $r = 0$. Suppose after r-th round of Step 1, $T(\mathbf{c})$ satisfies Property 1. In the $r+1$-th round, the deleting operation leaves blocks $\mathbf{b}_1$ and $\mathbf{b}_{\lceil \log n \rceil + 9 + \lceil \log k \rceil}$, which both contain $\mathbf{1}^{\lceil \log k \rceil + 5}$ as a subsequence. Hence $T(\mathbf{c})$ satisfies Property 1 after the deletion. In addition, all appended subsequences end with a $\mathbf{1}^{\lceil \log k \rceil + 5}$ pattern or a $\mathbf{1}^{\lceil \log k \rceil + 5}$ pattern followed by a 0 bit. Note that these appended subsequences are not deleted. Hence the index distance between two $\mathbf{1}^{\lceil \log k \rceil + 5}$ patterns in the appended bits $(T(\mathbf{c})_{n'+1}, \ldots, T(\mathbf{c})_{n+3k+3\lceil \log k \rceil + 15}$ is at most $R - 6k - 2\lceil \log k \rceil - 8 \leq B - \lceil \log k \rceil - 5$. Therefore, The sequence $T(\mathbf{c})$ satisfies Property 1 after the appending operation.

Next, we prove that $T(\mathbf{c})$ satisfies Property 2. According to the encoding procedure, for any $i \in [1, \min\{n', n + 3k + 3\lceil \log k \rceil + 16 - R\}]$, there exists some $j \in [i, i + R - 3k - \lceil \log k \rceil - 4]$, such that $(T(\mathbf{c})_m, T(\mathbf{c})_{m+1}, \ldots, T(\mathbf{c})_{m+\lceil \log k \rceil + 4}) \neq \mathbf{1}^{\lceil \log k \rceil + 5}$ for every $m \in [j, j + 3k - 1]$. Otherwise, the encoding does not stop. Note that the appended bits $(T(\mathbf{c})_{n'+1}, \ldots, T(\mathbf{c})_{n+3k+3\lceil \log k \rceil + 15})$ are not deleted. Hence for $i \in [n'+1, n+3k+3\lceil \log k \rceil + 16 - R]$, the interval $[i, i+R-1]$ contains the first $3k + \lceil \log k \rceil + 4$ bits $(0, T_2(\mathbf{b}_2))$ of some appended subsequence, where $\mathbf{b}_2 \in \{0,1\}^{3k + \lceil \log k \rceil + 4}$ contains the $\mathbf{1}^{\lceil \log k \rceil + 5}$ pattern. Let $[j, j + 3k + \lceil \log k \rceil + 3]$ be the indices of the $3k + \lceil \log k \rceil + 4$ bits $(0, T_2(\mathbf{b}_2))$ in $T(\mathbf{c})$. According to Proposition 3.6.4, the function $T_2(\mathbf{b}_2)$ does not contain the $\mathbf{1}^{\lceil \log k \rceil + 5}$ pattern. Hence $(T(\mathbf{c})_m, T(\mathbf{c})_{m+1}, \ldots, T(\mathbf{c})_{m+\lceil \log k \rceil + 4}) \neq \mathbf{1}^{\lceil \log k \rceil + 5}$ for every $m \in [j, j + 3k - 1]$. Therefore, any interval of length R contains a length $3k + \lceil \log k \rceil + 4$ subsequence with no $\mathbf{1}^{\lceil \log k \rceil + 5}$ pattern. Hence, the sequence $T(\mathbf{c})$ satisfies Property 2. According to Proposition 3.6.1, we conclude that $T(\mathbf{c})$ satisfies Property 1 and Property 2 and is k-*dense*.

Since n' decreases in the encoding procedure, the procedure terminates within $O(n)$ iterations. Each iteration takes $O(nk \log kR)$ time to search for the integer i satisfying the if condition and $O(\log nk^2 \log k)$ to compute the T_2 functions of the blocks. Hence the complexity is at most $O(n^2k^2 \log k(\log n))$. Therefore, the total complexity is $poly(n, k)$.

Finally we present the following decoding procedure that recovers $\mathbf{c}$ from $T(\mathbf{c})$, which follows a reverse procedure of the encoding.

1. **Initialization:** Let $\mathbf{c} = T(\mathbf{c})$ and go to Step 1.

2. **Step 1:** If $c_{n+3k+3\lceil \log k \rceil+15} = 0$, let i be the integer representation of $(c_{n+3k+2\lceil \log k \rceil+10-\lceil \log n \rceil}, c_{n+3k+2\lceil \log k \rceil+11-\lceil \log n \rceil}, \ldots, c_{n+3k+2\lceil \log k \rceil+9})$. Split $(c_{n+9k+5\lceil \log k \rceil+25-R}, \ldots, c_{n+3k+2\lceil \log k \rceil+9-\lceil \log n \rceil})$ into $\lceil \log n \rceil + 7 + \lceil \log k \rceil$ blocks $(\mathbf{b}'_1, \ldots, \mathbf{b}'_{\lceil \log n \rceil+7+\lceil \log k \rceil})$ of length $3k + \lceil \log k \rceil + 3$. Compute $\mathbf{b}_j = T_2^{-1}(\mathbf{b}'_j)$ for $j \in [1, \lceil \log n \rceil + 7 + \lceil \log k \rceil]$, where $T_2^{-1}(\mathbf{b}'_j)$ is defined in Proposition 3.6.4. Delete $(c_{n+9k+5\lceil \log k \rceil+24-R}, \ldots, c_{n+3k+3\lceil \log k \rceil+15})$ from $\mathbf{c}$ and insert $\mathbf{b}_1, \ldots, \mathbf{b}_{\lceil \log n \rceil+7+\lceil \log k \rceil}$ at location i of $\mathbf{c}$. Repeat. Else delete $(c_{n+2\lceil \log k \rceil+11}, \ldots, c_{n+3k+3\lceil \log k \rceil+15})$ and go to Step 2.

3. **Step 2:** Output $T_1^{-1}(\mathbf{c})$.

According to the encoding procedure, the inserted bits end with a 1 entry in the Initialization Step and with a 0 entry in Step 1. Note that the inserted bits are not deleted in the encoding procedure. Hence the decoding algorithm stops when an ending 1 entry is detected.

Similar to the proof of correctness of decoding in Proposition 3.6.3 and Proposition 3.6.4, the decoding procedure exactly reverses the series of operations in the encoding procedure. Therefore, the decoding procedure decodes the sequence $\mathbf{c}$ correctly.

3.7 Conclusion and Future Work

We construct a k-deletion correcting code with optimal order redundancy. Interesting open problems include finding complexity $O(n^{O(1)})$ encoding/decoding algorithms for our code, as well as constructing a systematic k-deletion correcting code with optimal redundancy.

Chapter 4

CORRECTING DELETIONS/INSERTIONS-GENERALIZATIONS

In this chapter, we develop a technique, called *syndrom compression*, based on an idea in our construction in Ch. 3. Using this technique, we construct a systematic binary k-deletion code with redundancy $4k\log n + o(\log n)$, which is asymptotically, the current best. Then, we extend our binary deletion correcting codes to more general settings, including non-binary cases. Note that DNA-based storage applications correspond to 4-ary cases.

4.1 Introduction

In the early 1960s, the problem of constructing codes for the deletion channel was introduced through the seminal works of Sellers and Levenshtein [82, 64]. Despite the inherent difficulty of coding for this channel, the initial results were promising. It was shown in [64] that the Varshmov-Tenengolts code can correct either a single deletion or a single insertion over a binary alphabet. Furthermore, [64] showed that $\log n + O(1)$ bits of redundancy are necessary for a single deletion code which implies that the Varshamov-Tenengolts code is nearly optimal. For the case where the channel deletes a burst of symbols has length 1 or 2, a nearly optimal (and very elegant) construction was provided in [63] by Levenshtein. In the early 1980s Tenengolts constructed a nearly optimal code for a single deletion over a non-binary alphabet based upon a clever mapping between a binary and a non-binary sequence [93].

Unfortunately, up until recently, progress was slow and results on the subject were limited. Some notable exceptions include the work of Schulman and Zuckerman [81] which showed the existence of efficient codes that are capable of correcting a constant fraction of deletions and the work of Helberg and Ferreira [47] which attempted to extend the Varshamov-Tenengolts code to handle more than a single deletion. Despite these efforts, up until a recent work by Brakensiek, Guruswami, and Zbarsky [12], no efficiently constructable codes were known that could correct even two deletions that required less than $\sqrt{n}$ bits of redundancy.

In [12], Brakensiek et al. constructed k deletion correcting codes that require

$O(k^2 \log k \log n)$ bits of redundancy. Several works quickly followed [12] that dramatically improved upon this result. Haeupler [41] gives an explicit systematic q-ary code construction which requires $\Theta(k\frac{\log^2 \frac{n}{k}}{\log q} + k)$ bits of redundancy. In [22], another construction was derived by Cheng et al., which is not systematic, but is order optimal in the sense that it requires $O(k \log n)$ bits of redundancy.

Independent of [22], in Ch. 3, we presented explicit constructions for k deletion correcting codes that have redundancy $8k \log n + o(\log n)$, which for the first time achieves asymptotically four times the Gilbert-Varshamov lower bound in the Levenshtein metric. In this chapter, we generalize one of the underlying techniques in Ch. 3, which we refer to as ***syndrome compression***. We show that using the syndrome compression technique, it is possible to construct codes for channels that are within a factor of two from the Gilbert-Varshamov lower bound in the Levenshtein metric. Note that the above multi-deletion correcting codes are not systematic except for the ones in [41] and [22], which have redundancy $\Theta(k \log^2 \frac{n}{k} + k)$. We furthermore present a systematic deletion correcting code and show that it achieves asymptotically twice the Gilbert-Varshamov lower bound, which is asymptotically the same amount of redundancy as our non-systematic construction, by using the syndrome compression technique. We then apply this technique to three different types of combinatorial deletion channels: (a) the binary adversarial deletion channel that can delete at most k symbols, (b) the binary adversarial deletion channel which can cause at most k bursts of deletions (each of length at most t_L), and (c) the non-binary adversarial deletion channel which can delete at most k symbols. We note that although our focus is on deletion channels, we believe the syndrome compression technique can yield meaningful results when applied to other types of channels (particular those that are traditionally considered difficult to code for, such as the deletion channel). In addition, we introduce new ideas for the systematic deletion code and the non-binary deletion correcting codes, as will be presented later, which might be of independent interest.

In the following, we highlight our contributions in variations of deletion channels.

Codes (non-systematic and systematic) correcting k deletions

We begin with a non-systematic k deletion correcting code construction that achieves asymptotically $4k \log n + O(\log n) + o(\log n)$ bits of redundancy, as a result of the syndrome compression technique.

Theorem 4.1.1. *Let k be a constant with respect to n. Then, there exists a* *non-systematic, efficiently encodable/decodable binary k-deletion correcting code of length n that requires at most $4k(1+\epsilon)\log n$ bits of redundancy where $\epsilon = O(1/k) + O(k^2 \log k/\log\log n)$ is a small number that depends on k and n. The encoding/decoding complexity is $O(n^{2k+1})$.*

In addition to the non-systematic deletion correcting codes, we are interested in systematic construction of deletion codes, motivated by the document exchange application [25], where Alice wishes to send a sequence X to Bob, who has the knowledge of a sequence Y that is within limited edit distance of X. The edit distance between X and Y is measured by the minimum number of deletions, insertions, or substitutions needed to change X into Y. It turns out that Alice can send the sequence X to Bob by transmitting a hash of X since Bob knows Y. Such a hash implies a systematic deletion correcting code and vice versa. Note that systematic deletion codes also has applications in storage systems with deletion errors, where concatenated code constructions are used.

The document exchange or systematic deletion code construction problem has been studied in both random and adversarial settings. For random settings, where a sequence can be recovered with high probability, document exchange algorithms with hash sizes $O(k\log^2 n)$ and $O(k\log^2 n\log^* n)$[1] were presented in [50] and [51], respectively. The results for randomized settings were improved to $O(k^2\log n)$ in [13] and to $O(k\log n)$ in [7]. In [44], a randomized systematic k-deletion correcting code with $O(k^2\log n)$ bits of redundancy was presented. For adversarial settings, where every sequence needs to be recovered in the worst case, the state of the art document exchange schemes [6, 22, 41] have hash size $O(k\log^2 n)$, which is bounded away from the lower bound $k\log n + o(\log n)$. Our next result gives a systematic deletion correcting codes with $4k\log n + o(\log n)$ bits of redundancy and thus a document exchange scheme with hash size $4k\log n + o(\log n)$. Note that this improves the redundancy of Theorem 4.1.1 by $O(\log n)$.

Theorem 4.1.2. *For any sequence $\mathbf{c} \in \{0,1\}^n$, there exists a hash function $Hash_k : \{0,1\}^n \to \{0,1\}^{4k\log n + o(\log n)}$, computable in $O(n^{2k+1})$ time, such that*

$$\{(\mathbf{c}, Hash_k(\mathbf{c})) : \mathbf{c} \in \{0,1\}^n\}$$

forms a k-deletion correcting code. The encoding/decoding complexity of the code is $O(n^{k+1})$.

[1]$\log^* n$ is the minimum number of times the logarithm needs to be iteratively applied before getting n to a result at most 1.

To the best of our knowledge, our results achieve asymptotically the best redundancy. The work in [90] proposed a non-systematic construction with $(4k-1)\log n + o(\log n)$ bits of redundancy, combining the syndrome compression technique and the VT code construction. The result is $\log n$ smaller than the result in this chapter. Yet, both results achieve the same order of redundancy asymptotically.

Codes correcting bursts of deletions

The results in this subsection pertain to the case where one wants to design codes capable of correcting bursts of deletions. The following presents a systematic code correcting a burst of at most k deletions.

Theorem 4.1.3. *Suppose k is a constant with respect to n. Then, there exists a systematic code of length n capable of correcting a burst of consecutive deletions of length at most k that requires at most $4\log n + o(\log n)$ bits of redundancy.*

Prior to this, codes capable of correcting a burst of consecutive deletions were provided in [35], where a construction is given that requires $\log k \log n +$ $O(k^2 \log k \log\log n)$ bits of redundancy so that our work represents a significant improvement. Furthermore, our construction is systematic whereas the one from [35] is not. Recently, the work of [60] presented a non-systematic code correcting a burst of k deletions with redundancy $\log n + k(k+1)/2 \log\log n + C$. We also consider the case where the deletions do not occur consecutively as well as the case where more than one burst occurs, and we provide constructions that improve upon prior art [80] in these cases as well. The following result shows that we can correct k bursts each of at most length t_L deletions, with redundancy at most $4k\log n + o(\log n)$.

Theorem 4.1.4. *Let k and t_L be two constant integers with respect to n. Then, there exists a systematic code of length n capable of correcting k bursts of consecutive deletions each of length at most t_L that requires at most $4\log n + o(\log n)$ bits of redundancy.*

Non-binary deletion correcting codes

Our next two results pertain to correcting deletions over non-binary alphabets, which has applications in DNA storage with alphabet size 4 and network transmission with packet loss. Despite the recent progress in binary deletion codes, the problem of constructing codes for the q-ary deletion channel has received significantly less attention. Tenengolts constructed a nearly optimal code for the case of a single deletion [93]. The main idea in [93] is to use a parity code to identify the symbol which was deleted and an associated Levenshtein code to determine the location of the deletion. For the case of multiple deletions, the Helberg codes [47], which were originally proposed for the binary deletion channel, were adapted and shown to produce non-binary deletion correcting codes [59]. The primary drawback to this class of codes is their low rate [59]. Even for the case of two deletions, the codes have rates that do not approach 1 as n becomes large. It was shown in [65] that the optimal redundancy of a q-ary k deletion code asymptotically falls between $k \log n + k \log q + o(\log qn)$ and $2k \log n + k \log q + o(\log qn)$.

We attempt to bring the existing results for non-binary codes closer to the results obtained in the binary domain.

Theorem 4.1.5. *Let k be a constant with respect to n and suppose that $q \leq \log n$. Then, there exists a systematic k-deletion code of message length n over an alphabet of size q that requires at most $4k \log n + o(\log n)$ bits of redundancy. When $\log n < q < n$, there exists a non-systematic k-deletion code of message length n over an alphabet of size q that requires $2k(1+\epsilon)(2\log n+\log q)+o(\log n)$ bits of redundancy.*

For the case where $q > n$, we use a different technique than syndrome compression that extends to the case where k is a constant fraction of n.

Theorem 4.1.6. *Let k be a constant with respect to n and suppose that $q \geq n$. Then, there exists a non-systematic, efficiently encodable/decodable k-deletion code of message length n over an alphabet of size q that requires at most $(30k + 1) \log q$ bits of redundancy.*

As will be explained in more detail in Sec. 4.7, the result stated in Theorem 4.1.6 also extends to the case where k is a constant fraction of code length n, when q is large enough. A similar case when k is a fraction of n and q is a polynomial of n was solved in [42].

We make use of two different approaches to obtain the results. For Theorem 4.1.5, we rely on the syndrome compression technique and some ideas that generalize the construction of [93]. For Theorem 4.1.6, we make use of results from repeat-free sequences from [28]. To the best of the authors' knowledge, the best known constructions of non-binary codes for the deletion channel can be found in [59] and so our results represent a significant improvement over existing work.

Organization

This chapter is organized as follows. In Sec. 4.2, we introduce the syndrome compression technique and prove its correctness. Sec. 4.3 describes our results for binary codes capable of correcting k deletions stated in Theorem 4.1.1 and Theorem 4.1.2. Sec. 4.5 provides constructions for the burst deletion channel. Sec.s 4.6 and 4.7 construct codes for the non-binary deletion channel. Finally Sec. 4.8 concludes this chapter.

4.2 Syndrome Compression

Although our attention is focused on deletion channels, syndrome compression is applicable to a wide variety of channels, and we begin our discussion by introducing the idea in its most general form. As will be described later, this general technique will result in explicit constructions for codes for various channels whose redundancy is roughly equal to twice the Gilbert-Varshamov bound (ignoring lower order terms).

For simplicity, in this section we limit our attention to the binary alphabet. Suppose the goal is to design a code $C \subseteq \{0,1\}^n$ such that for any vector $\mathbf{x} \in C$ and $\mathbf{y} \in \mathcal{B}(\mathbf{x})$, $\mathbf{y} \notin C$. We assume throughout this work, that $\mathbf{x} \notin \mathcal{B}(\mathbf{x})$. In other words, the set $\mathcal{B}(\mathbf{x})$ denotes the set of vectors distinct from $\mathbf{x}$ which are "confusable" with $\mathbf{x}$. It is assumed that for any $\mathbf{y} \in \mathcal{B}(\mathbf{x})$, both $\mathbf{x}$ and $\mathbf{y}$ have the same length. As an example, if we wish to design a code capable of correcting k deletions then $\mathcal{B}(\mathbf{x})$ would denote the set of vectors obtainable after deleting k symbols from $\mathbf{x}$ and then inserting k symbols into the resulting vector. Since $\log|\mathcal{B}(\mathbf{x})| \leq 2k\log n$ (See e.g., [12]) for the deletion channel, this implies that syndrome compression, when applied to the deletion channel, results in a code with roughly $4k\log n$ bits of redundancy.

In order to apply the general technique, we require a mapping:

$$f : \{0,1\}^m \to [[2^{R(m)}]] = \{0,1,\dots,2^{R(m)}-1\},$$

which takes as input any binary length m vector and it outputs an integer which is contained in the set $\{0,1,\dots,2^{R(m)}-1\}$. We assume the mapping f has the following two properties that are enumerated below:

1. **Confusability property**: For any $\mathbf{x} \in \{0,1\}^m$ and any $\mathbf{y} \in \mathcal{B}(\mathbf{x})$:

$$f(\mathbf{y}) \neq f(\mathbf{x}).$$

2. **Redundancy property**: $R(m) \leq o((\log\log m)\cdot\log m)$.

We will often refer to the mapping f as a *labeling*. When the meaning is clear, we will sometimes omit the argument m from $R(m)$ and just write R.

The main idea behind the construction is to start with the labeling f. If we define our code (using the usual techniques) to be comprised of the set of vectors that satisfy:

$$f(\mathbf{x}) \equiv a \bmod 2^R, \tag{4.1}$$

then using an averaging argument it follows that there exists a code that has redundancy roughly R bits. Syndrome compression is applicable to the case where it is known that $\log|\mathcal{B}(\mathbf{x})| << R \leq o((\log\log m) \cdot \log m)$, so that the R bits of redundancy are high relative to the minimum size of what the resulting code should be (since the resulting code should have at most $\log|\mathcal{B}(\mathbf{x})|$ bits of redundancy). In order to overcome this issue, we will reduce the modulus of (4.1), and thereby reducing the redundancy of the resulting code. In particular, for a fixed $\mathbf{x}$ we will find an integer $a < 2^R$ which for any $\mathbf{y} \in \mathcal{B}(\mathbf{x})$, it follows that

$$f(\mathbf{x}) \not\equiv f(\mathbf{y}) \bmod a. \tag{4.2}$$

Using a simple counting argument, we show in Lemma 4.2.2 that $\log|a| \lessapprox \log|\mathcal{B}(\mathbf{x})|$. As will be described in more details below, our approach will be to encode the information a along with the information $f(\mathbf{x}) \bmod a$. It is shown in Theorem 4.2.1 that the resulting construction results in a code C with the property that if $\mathbf{x} \in C$, then $\mathbf{y} \notin C$ for any $\mathbf{y} \in \mathcal{B}(\mathbf{x})$. Furthermore, since we need to encode the information a along with $f(\mathbf{x}) \bmod a$, it follows that the construction requires roughly $2\log|\mathcal{B}(\mathbf{x})|$ bits of redundancy.

Now we turn to a more formal treatment describing the construction along with the proofs of correctness. First, we cite a known result which bounds the number of divisors of a given number. This will be used to prove Lemma 4.2.2.

Lemma 4.2.1. *(c.f., [71]) For a positive integer $N \geq 3$, the number of divisors of N is upper bounded by*

$$2^{1.6 \cdot \frac{\ln N}{\ln\ln N}}.$$

Now we prove the key lemma which will be used to prove the main result in Theorem 4.2.1. For shorthand, for a vector $\mathbf{x} \in \{0,1\}^m$, let

$$\mathcal{D}(\mathbf{x}) = \Big\{ j : j > 0, j \Big| \big(f(\mathbf{x}) - f(\mathbf{y})\big), \mathbf{y} \in \mathcal{B}(\mathbf{x}) \Big\}.$$

Note that in order for Lemma 4.2.2 (stated below) to hold, we require both the confusability property and the redundancy property of f.

Lemma 4.2.2. *For any* $\mathbf{x} \in \{0,1\}^m$*,*

$$|\mathcal{D}(\mathbf{x})| \leq 2^{\log|\mathcal{B}(\mathbf{x})|+o(\log m)}.$$

Proof. We consider the set $\mathcal{D}(\mathbf{x})$. Note that for any element $j \in \mathcal{D}(\mathbf{x})$, there are two possibilities. Either

1. $j = |f(\mathbf{x}) - f(\mathbf{y})|$, or
2. j is a divisor of $|f(\mathbf{x}) - f(\mathbf{y})|$.

The number of possible choices for j such that 1) holds is equal to $|\mathcal{B}(\mathbf{x})|$, which follows from the confusability property of f. Furthermore, from Lemma 4.2.1, we know that for every number equal to $|f(\mathbf{x}) - f(\mathbf{y})|$ (for some $\mathbf{y} \in \mathcal{B}(\mathbf{x})$), there are at most $2^{1.6(\ln 2^R)/(\ln\ln 2^R)}$ divisors so that

$$|\mathcal{D}(\mathbf{x})| < |\mathcal{B}(\mathbf{x})| \cdot 2^{\frac{1.6}{\log e}\cdot\frac{R}{\ln\frac{R}{\log e}}},$$

which implies the result in the lemma statement since $R \leq o((\log\log m)\cdot\log m)$ from the redundancy property of f. □

Using the previous lemma, we can prove the following.

Theorem 4.2.1. *For any fixed* $\mathbf{x} \in \{0,1\}^m$*, there exists an integer* $a \leq 2^{\log|\mathcal{B}(\mathbf{x})|+o(\log m)}$ *such that for any* $\mathbf{y} \in \mathcal{B}(\mathbf{x})$*,*

$$f(\mathbf{x}) \not\equiv f(\mathbf{y}) \bmod a.$$

Proof. Assume that the parameter a is determined through a brute force search. In particular, we first attempt to set $a = 2$. If there exists a $\mathbf{y} \in \mathcal{B}(\mathbf{x})$ such that $2|(f(\mathbf{x}) - f(\mathbf{y}))$, then we set $a = 3$. It is straightforward to see that since the set $\mathcal{D}(\mathbf{x}) = \left\{j : j > 0, j\middle|(f(\mathbf{x}) - f(\mathbf{y}))\right\}$ has cardinality at most $2^{\log|\mathcal{B}(\mathbf{x})|+o(\log m)}$ there exists an a that satisfies the theorem statement. □

In light of Theorem 4.2.1, we now describe at a high level the encoding process for syndrome compression. Let $\mathbf{u} \in \{0,1\}^n$ be a binary sequence of length n and suppose a is such that $f(\mathbf{u}) \not\equiv f(\mathbf{y}) \bmod a$ for any $\mathbf{y} \in \mathcal{B}(\mathbf{u})$ from Theorem 4.2.1. Next, we append a vector $\mathbf{r} \in \{0,1\}^r$ to $\mathbf{u}$. The vector $\mathbf{r}$ will contain the information

1. a,

2. $f(\mathbf{u}) \bmod a$.

The codewords will be of the form

$$\mathbf{x} = (\mathbf{u}, \mathbf{r}) \in \{0,1\}^{n+r}. \tag{4.3}$$

For cases where $\mathbf{u}$ can be ANY vector of length n, the resulting construction is systematic. Note that according to Theorem 4.2.1, our approach requires a brute force search for the parameter a which has time-complexity at most $O(n^{2k})$. For the case where $k = O(1)$, the technique admits a polynomial-time encoding/decoding algorithm (since the decoding may also be performed using standard brute force methods). However, if k grows as a function of n, brute-force encoding algorithms do not result in polynomial-time complexity, and so our focus for the remainder of this work will be on the setup where k is constant with respect to n. We will discuss this issue again in Sec. 4.8. The next section applies this technique to the case where one wants to design codes that correct multiple deletions.

4.3 Non-Systematic Deletion Correcting Codes

In this section, we give a non-systematic construction which requires $4k(1+\epsilon)\log n$ bits of redundancy for a code capable of correcting k deletions where $\epsilon << 1$ given that k, n are large enough and $k \leq O((\log\log n)^{\frac{1}{2}-\delta})$. The basic idea will be to combine the syndrome compression technique described in Sec. 4.2 with the labeling from [12]. We note that although the resulting construction is not systematic, the codewords can be partitioned into two parts similar to what we generically described in the previous section. In particular, for the case where no errors occur, the original information sequence can be re-constructed by taking the xor of the first n bits of a codeword with a string which is described by the last r bits of the codeword, the form of which is given by (4.3). Furthermore we will show that if we wish to correct k deletions with high probability and the messages are iid uniform, a problem which was considered in [44], our construction yields a systematic encoding while requiring less redundancy than the codes from [44]. While both the systematic codes in [44] and this chapter requires $n^{O(k)}$ encoding/decoding complexity, our systematic codes have $4k\log n + o(\log n)$-bit redundancy, compared to the $k(k+1)\log n$-bit redundancy in [44]. Similar probabilistic decoding settings were also considered in construction of polar codes [92] for deletion correction, where k is a fraction of n and the parameter of interest is code rate instead of code redundancy. In the next section, we will further construct a systematic k-deletion correcting code under adversarial settings, which has asymptotically the same redundancy $4k \log n + o(\log n)$ as the code under probabilistic settings.

We begin with some results and definitions from [12]. The next lemma describes the labeling, which will be used throughout this section. In the following, we refer to $\mathcal{M}(m,k) \subseteq \{0,1\}^m$ as the set of k-mixed strings of length m. More precisely,

$\mathcal{M}(m,k) = \{\mathbf{x} \in \{0,1\}^m :$ For integers $\ell = \lceil \log k + \log\log(k+1) + 5 \rceil$ and $d = O(k(\log k)^2 \log m)$ and for any string $\boldsymbol{p} \in \{0,1\}^\ell$, every substring of consecutive d bits in $\mathbf{x}$ contains $\boldsymbol{p}$ as a substring.$\}$

For a vector $\mathbf{x}$, the set $\mathcal{B}_k(\mathbf{x})$ is the set of vectors obtainable after deleting any k symbols from $\mathbf{x}$ and then inserting k symbols into the resulting vector. Recall from the previous section that $|\mathcal{B}_k(\mathbf{x})| \leq m^{2k}$.

Lemma 4.3.1. *Fix an integer $k \geq 2$. Then for all large m, there exists $R(m) = O(k^2 \log k \log m)$ and a hash function $f_k : \mathcal{M}(m,k) \to \{0,1\}^{R(m)}$ so that for any distinct $\mathbf{x}, \mathbf{y} \in \mathcal{M}(m,k)$,*

$$f_k(\mathbf{x}) \neq f_k(\mathbf{y}),$$

if $\mathbf{y} \in \mathcal{B}_k(\mathbf{x})$.

Note from the previous lemma that if we want to use the mapping f_k, we need to restrict our sequences to be from the set of k-mixed strings. In order to transform arbitrary information sequences into k-mixed strings from the set $\mathcal{M}(m,k)$,we will require the following result, which is proven in Appendix 4.9 using similar techniques from [22].

Lemma 4.3.2. *There exists a $poly(m)$ time algorithm for a seed s with length $O(\log m)$ such that for any $\mathbf{u} \in \mathbb{F}_2^m$, $\mathbf{u} + g(s) \in \mathcal{M}(m,k)$.*

We'll also make use of the following code from [22].

Lemma 4.3.3. *For any m, k, there exists an encoder $\mathcal{E}_k$ for a code with m bits of information and $O(k \log m)$ bits of redundancy capable of correcting k deletions.*

In light of the previous two lemmas we can now present a construction for a k-deletion correcting code, which we now describe by means of our encoding procedure. The input to our encoder is an information sequence $\mathbf{u} \in \{0,1\}^n$ and the output is a codeword $\mathbf{x} \in \{0,1\}^{n+4k\log n + O(\log n)}$. In the following, let $\mathbf{0}_m$ denote the all-zeros vector of length m and similarly let $\mathbf{1}_m$ denote the all-ones vector of length m.

1. Let $\mathbf{u} \in \{0,1\}^n$ and suppose s is such that $\mathbf{u}_T = \mathbf{u} + g(s) \in \mathcal{M}(n,k)$.
2. Suppose $a \in [[n^{2k}]]$ is such that $f_k(\mathbf{u}_T) \not\equiv f_k(\mathbf{y}) \bmod a$ for any $\mathbf{y} \in \mathcal{B}_k(\mathbf{u}_T)$.
3. Then,

$$\mathbf{x} = \Big(\mathbf{u}_T, \mathbf{0}_{k+1}, \mathbf{1}_{k+1}, \mathcal{E}_k\big(s, a, f_k(\mathbf{u}_T) \bmod a\big)\Big) \in \{0,1\}^n.$$

We have the following theorem, which proves Theorem 4.1.1.

Theorem 4.3.1. *Suppose z is the result of at most k deletions occurring to $\mathbf{x}$. Then, it is possible to uniquely recover $\mathbf{x}$ provided $\mathbf{z}$.*

Proof. To prove the result, we show that it is possible to recover $\mathbf{u}$ from $\mathbf{z}$. First, notice that if $\mathbf{x}$ is transmitted and $\mathbf{z}$ is received, then $z_{n+1} = 0$. This is because there are at most k deletions which can occur to $\mathbf{x}$ (resulting in $\mathbf{z}$) and so the run of zeros which immediately follows $\mathbf{u}_T$ in $\mathbf{x}$ must also appear in $\mathbf{z}$ and, by similar logic, the first 1 which follows z_{n+1} corresponds to the first run of ones in $\mathbf{x}$ after $\mathbf{u}_T$. Suppose the first 1 that occurs after z_{n+1} appears in position p where $k' = n + (k+2) - p$. Then, k' symbols have been deleted from the first $n + (k+1)$ positions in $\mathbf{x}$ resulting in $\mathbf{z}$. This implies that $k'' = |\mathbf{x}| - |\mathbf{z}| - k'$ deletions have occurred in the final $(k+1) + |\mathcal{E}_k\big(s, a, f_k(\mathbf{u}_T) \bmod a\big)|$ bits of $\mathbf{x}$. Therefore, the following holds:

1. The last $|\mathcal{E}_k\big(s, a, f_k(\mathbf{u}_T) \bmod a\big)| - k''$ bits of $\mathbf{z}$ can be obtained by deleting k'' bits from $\mathcal{E}_k\big(s, a, f_k(\mathbf{u}_T) \bmod a\big)$.
2. The first $n - k'$ bits of $\mathbf{z}$ can be obtained by deleting k' bits from $\mathbf{u}_T$.

Thus, we can recover $\big(s, a, f_k(\mathbf{u}_T) \bmod a\big)$ from the last $|\mathcal{E}_k\big(s, a, f_k(\mathbf{u}_T) \bmod a\big)| - k''$ bits of $\mathbf{z}$ and the vector $\mathbf{u}_T$ can be obtained from the first $n - k'$ bits of $\mathbf{z}$ given $a, f_k(\mathbf{u}_T) \bmod a$. From s, we can generate $g(s)$ and finally we can recover $\mathbf{u}$. □

Next, we adapt our encoding to handle the setup where $\mathbf{u} \in \{0,1\}^n$ is a uniform iid message, which is the setup considered in [44]. The input to our encoder is an information sequence $\mathbf{u} \in \{0,1\}^n$ and the output is a systematic codeword $\mathbf{x} \in \{0,1\}^{n+4k\log n + o(\log n) + O(k\log(4k\log n))}$

1. Let $\mathbf{u} \in \{0,1\}^n$.
2. Suppose $a \in [[n^{2k}]]$ is such that $f_k(\mathbf{u}) \not\equiv f_k(\mathbf{y}) \bmod a$ for any $\mathbf{y} \in \mathcal{B}_k(\mathbf{u})$.
3. Then,

$$\mathbf{x} = \Big(\mathbf{u}, \mathbf{0}_{k+1}, \mathbf{1}_{k+1}, \mathcal{E}_k\big(a, f_k(\mathbf{u}) \bmod a\big)\Big) \in \{0,1\}^n.$$

The next corollary follows immediately Appendix 4.9 and the proofs of Claim 4.9.1, Lemma 4.3.2, and Theorem 4.3.1. A proof is included at the end of Appendix 4.9.

Corollary 4.3.1. *For a uniform iid message* $\mathbf{u} \in \{0,1\}^n$*, the code*

$$C = \Big\{\mathbf{x} = \Big(\mathbf{u}, \mathbf{0}_{k+1}, \mathbf{1}_{k+1}, \mathcal{E}_k\big(a, f_d(\mathbf{u}) \bmod a\big)\Big) \in \{0,1\}^n\Big\}$$

can correct k deletions with probability at least $1 - 1/poly(n)$ for sufficiently large n.

4.4 Systematic Deletion Correcting Codes

In this section, we provide systematic k deletion correcting codes that achieve $4k \log n + o(\log n)$ bits of redundancy, improving that of the non-systematic construction by $O(\log n)$. Similar to the non-systematic code construction, we use the syndrome compression technique. However, to apply the technique, we need a deletion correcting hash for every length n binary sequence with size $o((\log\log n)\cdot\log n)$ to start with. Note that none of the existing codes or document exchange schemes achieve this size.

The key ideas for obtaining a deletion correcting hash generalize the techniques in our previous work [86]. They are sketched as follows: (i) generalizing the VT-construction to correct deletions and substitutions for constrained sequences, (ii) decomposing a sequence into multiple versions of it with different resolution levels such that the version with the lowest resolution is a constrained sequence.

In [86] we generalized the VT construction and proved that the higher order parities $\sum_{i=1}^{n} c_i i^e \bmod kn^e, e \in [0, 2k] = \{0, 1, \ldots, 2k\}$ are a k-deletion correcting hash for sequences $\mathbf{c}$, in which any two 1 entries are separated by at least $k-1$ 0 entries. Motivated by this observation, we define the $\mathbf{u}$-indicator vector $\mathbb{1}_{\mathbf{u}}(\mathbf{c}) \in \{0,1\}^n$ of $\mathbf{c}$ element-wise for binary sequences $\mathbf{c}$ and $\mathbf{u}$. Let n and $\ell \le n$ be the length of $\mathbf{c}$ and $\mathbf{u}$ respectively, define $\mathbb{1}_{\mathbf{u}}(\mathbf{c})$ by

$$\mathbb{1}_{\mathbf{u}}(\mathbf{c})_i = \begin{cases} 1, & \text{if } i \le n - \ell + 1 \text{ and } (c_i, \ldots, c_{i+\ell-1}) = \mathbf{u}, \\ 0, & \text{else.} \end{cases}$$

The special cases of $\mathbb{1}_{\mathbf{u}}(\mathbf{c})$ when $\mathbf{u} = (0,1)$ or $\mathbf{u} = (1,0)$ were considered in [88], where a sequence $\mathbf{c}$ is decomposed into a $(1,0)$-indicator vector and a $(0,1)$-indicator vector for $k = 2$. In this chapter we generalize this decomposition for $k > 2$. We iteratively generate k versions of $\mathbf{c}$ with different levels of resolution. Let $I_1(\mathbf{c}) = \mathbf{c}$ and

$$I_{w+1}(\mathbf{c}) = \mathbb{1}_{(1,\mathbf{0}^w)}(I_w(\mathbf{c})),$$

for $w \in [k-1] \triangleq \{1, \ldots, k-1\}$, where $(1, \mathbf{0}^w)$ is a sequence of a 1 entry followed by w 0 entries. The 1 entries of $I_{w+1}(\mathbf{c})$ are also 1 entries of $I_w(\mathbf{c})$, $w \in [k-1]$.

Example 4.4.1. *For $k = 3$ and $\mathbf{c} = 10010110100$, we have that $I_1(\mathbf{c}) = \mathbb{1}_1(\mathbf{c}) = 10010110100$, $I_2(\mathbf{c}) = \mathbb{1}_{(1,0)}(I_1(\mathbf{c})) = 10010010100$, and that $I_3(\mathbf{c}) = \mathbb{1}_{(1,0,0)}(I_2(\mathbf{c})) = 10010000100$.*

The nice properties of $I_w(\mathbf{c})$ are as follows. (1) Any two 1 entries in $I_k(\mathbf{c})$ are separated by at least $k-1$ 0 entries. (2) The vector $I_w(\mathbf{c})$ is highly constrained when $I_{w+1}(\mathbf{c})$ is known, as will be discussed in the proof of Lemma 4.4.3. The first property guarantees that $I_k(\mathbf{c})$ can be protected using higher order parities. The second property enables a successive decoding algorithm.

We first investigate the effect on $I_w(\mathbf{c})$ caused by k deletions in $\mathbf{c}$. For a non-negative integer i, let $\mathcal{B}_{k,i}(\mathbf{c})$ be the set of sequences that can be obtained after deleting k bits and substituting i bits in $\mathbf{c}$. The following lemma gives an upper bound on the number of deletions and substitutions in $I_w(\mathbf{c})$, caused by k deletions in $\mathbf{c}$.

Lemma 4.4.1. *For sequences $\mathbf{c}, \mathbf{c}' \in \{0,1\}^n$, if there exists a subsequence $\mathbf{d} \in \mathcal{B}_k(\mathbf{c}) \cap \mathcal{B}_k(\mathbf{c}')$, then $I_w(\mathbf{d}) \in \mathcal{B}_{k,k(w-1)}(I_w(\mathbf{c})) \cap \mathcal{B}_{k,k(w-1)}(I_w(\mathbf{c}'))$ for $w \in [k]$.*

Proof. We show that a deletion in $\mathbf{c}$ causes at most a deletion and $w-1$ substitutions in $I_w(\mathbf{c})$. To this end, we prove in the following that the deletion of c_i causes a deletion in $I_w(\mathbf{c})$ and multiple substitutions that occur within interval $[i - w(w-1)/2, i-1]$ in $I_w(\mathbf{c})$. Since for any sequence $\mathbf{c} \in \{0,1\}^m$, we have $I_w(\mathbf{c}) \in \mathcal{R}_{w,m}$ and $I_w(\mathbf{c}) \in \mathcal{R}_{w,m-1}$ before and after deleting c_i in $\mathbf{c}$, respectively, it follows that there are at most $(w-1)/2$ substitution errors that change 1 entries to 0 entries and 0 entries to 1 entries respectively. Hence, a deletion in $\mathbf{c}$ causes a deletion and at most $w-1$ substitutions in $I_w(\mathbf{c})$. Since $\mathbf{d} \in \mathcal{B}_k(\mathbf{c})$, it follows that $I_w(\mathbf{d}) \in \mathcal{B}_{k,k(w-1)}(\mathbf{c})$. Similarly, we have that $I_w(\mathbf{d}) \in \mathcal{B}_{k,k(w-1)}(\mathbf{c}')$.

We show how a deletion or substitution that occurs at $I_j(\mathbf{c})_{i_j}$, $i_j \in [n]$, affects $I_{j+1}(\mathbf{c})$, $j \in [1, w-1]$. Let $i^*_j = \max_{\ell < i_j, I_j(\mathbf{c})_\ell = 1} \ell$ be the index of the last 1 entry in $I_j(\mathbf{c})$ before $I_j(\mathbf{c})_{i_j}$, and $i^{**}_j = \min_{\ell > i_j, I_j(\mathbf{c})_\ell = 1} \ell$ be the index of the first 1 entry in $I_j(\mathbf{c})$ after $I_j(\mathbf{c})_{i_j}$, where it is assumed that $I_j(\mathbf{c})_\ell = 1$ when $\ell = 0$ or $\ell = n+1$.

1. If $i^*_j \le i_j - j - 1$, then the deletion or substitution of $I_j(\mathbf{c})_{i_j}$ in $I_j(\mathbf{c})$ causes the deletion or substitution of $I_{j+1}(\mathbf{c})_{i_j}$ in $I_{j+1}(\mathbf{c})$ respectively.
2. If $i^*_j \ge i_j - j$ and $I_j(\mathbf{c})_{i_j} = 1$, then the deletion or substitution of $I_j(\mathbf{c})_{i_j}$ in $I_j(\mathbf{c})$ causes a deletion or at most a substitution of $I_{j+1}(\mathbf{c})_{i_j}$ in $I_{j+1}(\mathbf{c})$, respectively.
3. If $i^*_j \ge i_j - j$ and $I_j(\mathbf{c})_{i_j} = 0$, then the substitution of $I_j(\mathbf{c})_{i_j}$ in $I_j(\mathbf{c})$ causes at most two substitutions of $I_{j+1}(\mathbf{c})_{i^*_j}$ and $I_{j+1}(\mathbf{c})_{i_j}$ in $I_{j+1}(\mathbf{c})$. The deletion of $I_j(\mathbf{c})_{i_j}$ in $I_j(\mathbf{c})$ causes the deletion of $I_{j+1}(\mathbf{c})_{i_j}$ and the substitution of $I_{j+1}(\mathbf{c})_{i^*_j}$ in $I_{j+1}(\mathbf{c})$, when $i^{**}_j - i^*_j \le j+1$, and causes only the deletion of $I_{j+1}(\mathbf{c})_{i_j}$ in $I_{j+1}(\mathbf{c})$ when $i^{**}_j - i^*_j \ge j+2$.

In all cases above, a deletion of $I_j(\mathbf{c})_{i_j}$ in $I_j(\mathbf{c})$ causes a deletion and at most one substitution that occurs in the range $[i_j - j, i_j] = \{i_j - j, i_j - j + 1, \ldots, i_j\}$, and a substitution of $I_j(\mathbf{c})_{i_j}$ in $I_j(\mathbf{c})$ causes at most two substitutions that occur in the range $[i_j - j, i_j]$, for $j \in [w-1]$. Using induction on j we can prove that the deletion of $c_i = I_1(\mathbf{c})_i$ causes one deletion and substitutions that occur in the range $[i - (1 + \ldots + w - 1), i - 1] = [i - w(w-1)/2, i-1]$. □

For any integer m, let $\mathcal{R}_{k,m}$ be the set of length m sequences where any two 1 entries are separated by at least $k-1$ 0 entries. Given the upper bounds of deletions and substitutions in Lemma 4.4.1, we next show that the generalization of VT constraints $f(\mathbf{x})$ can be used to correct these deletions and substitutions for constrained sequences $\mathbf{x} \in \mathcal{R}_{k,n}$. We define the generalized VT constraint as follows. Define the integer vectors

$$\mathbf{m}^{(e)} \triangleq (1^e, 1^e + 2^e, \ldots, \sum_{j=1}^{n} j^e),$$

for $e \in [0, 2k^2]$. For any sequence $\mathbf{c} \in \{0,1\}^n$, let $f(\mathbf{c})$ be a $2k^2+1$-dimensional vector given by

$$f(\mathbf{c})_e = \mathbf{c} \cdot \mathbf{m}^{(e)} \bmod k^2 n^{e+1},$$

for $e \in [0, 2k^2]$.

Lemma 4.4.2. *For any two sequences* $\mathbf{x}, \mathbf{x}' \in \{0,1\}^n$*, if there exists a sequence* $\mathbf{z} \in \mathcal{R}_{k,n-k}$ *satisfying* $\mathbf{z} \in \mathcal{B}_{k,k(k-1)}(\mathbf{x}) \cap \mathcal{B}_{k,k(k-1)}(\mathbf{x}')$*, we have that* $f(\mathbf{x}) \neq f(\mathbf{x}')$*.*

Proof. Note that by Lemma 4.4.1 we have that $I_k(\mathbf{z}) \in \mathcal{R}_{k,n-k}$ and that $I_k(\mathbf{z}) \in \mathcal{B}_{k,k(k-1)}(I_k(\mathbf{x})) \cap \mathcal{B}_{k,k(k-1)}(I_k(\mathbf{x}'))$. By virtue of this lemma, the redundancy $f(I_k(\mathbf{c}))$ can be used to correct $I_k(\mathbf{c})$. The proof of Lemma 4.4.2 follows similar arguments to those in [86], which show that any sequence in $\mathcal{R}_{k,n}$ can be protected from k deletions by using higher oder parities. Here slight changes are made in order to deal with additional substitutions.

Let $\boldsymbol{\delta} = \{\delta_1, \ldots, \delta_k\} \subset [n]$ be a set of deletion indices and $\boldsymbol{\sigma} = \{\sigma_1, \ldots, \sigma_{k(k-1)}\} \subset [n]$ be a set of substitution indices, such that deleting bits $(x_i : i \in \boldsymbol{\delta})$ and substituting bits $(x_i : i \in \boldsymbol{\sigma})$ in $\mathbf{x}$ result in $\mathbf{z}$. Similarly, let $\boldsymbol{\delta}' = \{\delta_1', \ldots, \delta_k'\} \subset [n]$ and $\boldsymbol{\sigma} = \{\sigma_1, \ldots, \sigma_{k(k-1)}\} \subset [n]$ be two sets such that deleting bits $(x_i' : i \in \boldsymbol{\delta'})$ and substituting bits $(x_i' : i \in \boldsymbol{\sigma'})$ in $\mathbf{x}'$ result in $\mathbf{z}$. Let $\mathbf{y}$ and $\mathbf{y}'$ be the sequences obtained by substituting bits $(x_i : i \in \boldsymbol{\sigma})$ in $\mathbf{x}$ and substituting bits $(x_i' : i \in \boldsymbol{\sigma'})$ in $\mathbf{x}'$, respectively. Then we have that $\mathbf{z} \in \mathcal{B}_k(\mathbf{y}) \cap \mathcal{B}_k(\mathbf{y}')$. Moreover, the sequence $\mathbf{z}$ can be obtained by deleting $(y_i : i \in \boldsymbol{\delta})$ from $\mathbf{y}$ or deleting $(y_i' : i \in \boldsymbol{\delta'})$ from $\mathbf{y}'$.

We now compute the difference $\mathbf{x} \cdot \mathbf{m}^{(e)} - \mathbf{x}' \cdot \mathbf{m}^{(e)}$ and show that it cannot be 0 for all $e \in [0, 2k^2]$ unless $\mathbf{x} = \mathbf{x}'$. Following the same steps as in [86], let $\boldsymbol{\Delta} = \{i : y_i = 1\}$ and $\boldsymbol{\Delta'} = \{i : y_i' = 1\}$ be indices of 1 entries in $\mathbf{y}$ and $\mathbf{y}'$ respectively. Let $S_1 = \boldsymbol{\Delta} \cap \boldsymbol{\delta}$ and $S_2 = \boldsymbol{\Delta'} \cap \boldsymbol{\delta'}$be indices of the 1 entries, after deleting which in $\mathbf{y}$ and $\mathbf{y}'$, respectively, we have $\mathbf{z}$. Then, the sets $S_1^c = [n] \backslash S_1$ and $S_2^c = [n] \backslash S_2$ are indices of the 1 entries that are not deleted in $\mathbf{y}$ and $\mathbf{y}'$ respectively. We have that

$$\begin{aligned}
\mathbf{y} \cdot \mathbf{m}^{(e)} - \mathbf{y}' \cdot \mathbf{m}^{(e)} &= \sum_{\ell \in \boldsymbol{\Delta}} \Big(\sum_{i=1}^{\ell} i^e\Big) - \sum_{\ell \in \boldsymbol{\Delta'}} \Big(\sum_{i=1}^{\ell} i^e\Big) \\
&= \sum_{i=1}^{n} (|S_1 \cap [i,n]| + |S_1^c \cap [i,n]| \\
&\quad - |S_2 \cap [i,n]| - |S_2^c \cap [i,n]|) i^e. \qquad (4.4)
\end{aligned}$$

Sort all elements in sets $\boldsymbol{\delta}, \boldsymbol{\delta'}, \boldsymbol{\sigma}$, and $\boldsymbol{\sigma'}$ by $p_1 \leq p_2 \leq \ldots \leq p_{2k^2}$. Let $p_0 = 0$ and $p_{2k^2+1} = n$. The sets S_1 and S_2 satisfy the following properties, the proof of which follows the same steps as in [86].

1. $-1 \leq |S_1^c \cap [i,n]| - |S_2^c \cap [i,n]| \leq 1$ for $i \in [n]$.

2. For each interval $(p_j, p_{j+1}] \triangleq \{p_j + 1, \dots, p_{j+1}\}$, $j \in [0, 2k^2]$, we have either $|S_1^c \cap [i,n]| - |S_2^c \cap [i,n]| \leq 0$ for all $i \in (p_j, p_{j+1}]$ or $|S_1^c \cap [i,n]| - |S_2^c \cap [i,n]| \geq 0$ for all $i \in (p_j, p_{j+1}]$.

Let the sets $S_3 = \Delta \cap \boldsymbol{\sigma} = \{i : y_i = 1, i \in \boldsymbol{\sigma}\}$ and $S_4 = \Delta' \cap \boldsymbol{\sigma'} = \{i : y'_i = 1, i \in \boldsymbol{\sigma'}\}$ be the indices of substitutions that flip 0 bits in $\mathbf{x}$ and $\mathbf{x}'$, in order to get $\mathbf{y}$ and $\mathbf{y}'$ respectively. Let $S_5 = \boldsymbol{\sigma} \backslash S_3 = \{i : y_i = 0, i \in \boldsymbol{\sigma}\}$, and $S_6 = \boldsymbol{\sigma} \backslash S_4 = \{i : y_i = 0, i \in \boldsymbol{\sigma'}\}$ be the indices of substitutions that flip 1 bits in $\mathbf{x}$ and $\mathbf{x}'$, to get $\mathbf{y}$ and $\mathbf{y}'$, respectively. Then,

$$\begin{aligned}
&\mathbf{x} \cdot \mathbf{m}^{(e)} - \mathbf{x}' \cdot \mathbf{m}^{(e)} \\
&= \mathbf{y} \cdot \mathbf{m}^{(e)} - \mathbf{y}' \cdot \mathbf{m}^{(e)} + \sum_{\ell \in S_5} \mathbf{m}_\ell^{(e)} - \sum_{\ell \in S_6} \mathbf{m}_\ell^{(e)} \\
&\quad - (\sum_{\ell \in S_3} \mathbf{m}_\ell^{(e)} - \sum_{\ell \in S_4} \mathbf{m}_\ell^{(e)}) \\
&= \sum_{j=0}^{2k^2} \sum_{i=p_j+1}^{p_{j+1}} (|S_1^c \cap [i,n]| - |S_2^c \cap [i,n]| + k_i) i^e,
\end{aligned} \tag{4.5}$$

where $k_i = |S_1 \cap [i,n]| - |S_2 \cap [i,n]| + |S_5 \cap [i,n]| - |S_3 \cap [i,n]| + |S_4 \cap [i,n]| - |S_6 \cap [i,n]|$ for $i \in [n]$. Note that for any interval $(p_j, p_{j+1}]$, $j \in [0, 2k^2]$, the number k_i is constant for all $i \in (p_j, p_{j+1}]$. Let $s_i = |S_1^c \cap [i,n]| - |S_2^c \cap [i,n]| + k_i$, then it follows from Property (1) and Property (2) that s_i is either negative or non-negative for all $i \in (p_j, p_{j+1}]$ for each $j \in [0, 2k^2]$.

Next, we show that $\mathbf{x} \cdot \mathbf{m}^{(e)} - \mathbf{x}' \cdot \mathbf{m}^{(e)}$ cannot be zero for all $e \in [0, 2k^2]$ when $\mathbf{x} \neq \mathbf{x}'$. Otherwise, define a vector $\mathbf{v} = (v_0, \dots, v_{2k^2}) \in \{-1, 1\}^{2k^2+1}$ by

$$v_j = \begin{cases} -1, & \text{if } s_i < 0 \text{ for some } i \in (p_j, p_{j+1}] \\ 1, & \text{else.} \end{cases},$$

and a $(2k^2+1) \times (2k^2+1)$ matrix A by $A_{e,j} = \sum_{i=p_j+1}^{p_j} |s_i| i^e$ for $e, j \in [0, 2k^2]$. Then according to Eq. (4.5), we have that $\sum_{j=0}^{2k^2} A_{e,j} v_j = 0$, $e \in [0, 2k^2]$, if $\mathbf{x} \cdot \mathbf{m}^{(e)} - \mathbf{x}' \cdot \mathbf{m}^{(e)} = 0$ for all $e \in [0, 2k^2]$. This implies the linear equation $A\mathbf{v} = 0$ has a solution $\mathbf{v}$ with no 0 entry. The remaining steps are the same as in [86]. Let $j_1, \dots, j_Q$ be the indices of non-zero columns of A, and B be the submatrix of A by selecting the intersection of the first Q rows and the non-zero columns of A. Then the linear equation $B(v_{j_1}, \dots, v_{j_Q}) = 0$ has a non-zero solution, which is impossible since by multi-linearity of the determinant, we can prove that the determinant $|B| > 0$.

Therefore, $\mathbf{x} \cdot \mathbf{m}^{(e)} - \mathbf{x}' \cdot \mathbf{m}^{(e)} = 0$ for $e \in [0, 2k^2]$ only when A is a zero matrix, which implies that

$$\begin{aligned} s_i &= |S_1^c \cap [i,n]| - |S_2^c \cap [i,n]| + k_i \\ &= |\{i : x_i = 1\} \cap [i,n]| - |\{i : x_i' = 1\} \cap [i,n]| = 0, \end{aligned}$$

for $i \in [n]$. Then, $\{i : x_i = 1\} = \{i : x_i' = 1\}$ and thus $\mathbf{x} = \mathbf{x}'$.

Finally, we show that if $f(\mathbf{c}) = f(\mathbf{c}')$, then $\mathbf{x} \cdot \mathbf{m}^{(e)} - \mathbf{x}' \cdot \mathbf{m}^{(e)} = 0$ for $e \in [0, 2k^2]$. Since $\mathbf{z} \in \mathcal{B}_{k,k(k-1)}(\mathbf{x}) \cap \mathcal{B}_{k,k(k-1)}(\mathbf{x}')$, it follows that $(z_i, \ldots, z_{n-k}) \in \mathcal{B}_{k,k(k-1)}((x_i, \ldots, x_n)) \cap \mathcal{B}_{k,k(k-1)}((x_i', \ldots, x_n'))$ for $i \in [n-k]$. Hence, we have that $-k^2 \leq |\{i : x_i = 1\} \cap [i,n]| - |\{i : x_i' = 1\} \cap [i,n]| \leq k^2$, and that

$$|\mathbf{x} \cdot \mathbf{m}^{(e)} - \mathbf{x}' \cdot \mathbf{m}^{(e)}| < k^2 n^{e+1}.$$

Therefore, if $f(\mathbf{c}) = f(\mathbf{c}')$, we have that $\mathbf{x} \cdot \mathbf{m}^{(e)} - \mathbf{x}' \cdot \mathbf{m}^{(e)} \equiv 0 \bmod k^2 n^{n+1}$, which implies that $\mathbf{x} \cdot \mathbf{m}^{(e)} - \mathbf{x}' \cdot \mathbf{m}^{(e)} = 0$, for $e \in [0, 2k^2]$. □

After protecting $I_k(\mathbf{c})$ in Lemma 4.4.2, the following lemma provides a hash function that recovers $I_w(\mathbf{c})$, $w \in [k-1]$, from deletions and substitutions, the numbers of which are upper bounded in Lemma 4.4.1, when $I_{w+1}(\mathbf{c})$ is known.

Lemma 4.4.3. *For any two sequences $\mathbf{c}, \mathbf{c}' \in \{0,1\}^n$, if there exists a sequence $\mathbf{d}$ satisfying $\mathbf{d} \in \mathcal{B}_k(\mathbf{x}) \cap \mathcal{B}_k(\mathbf{c}')$, then there exists a hash function $H_w : \{0,1\}^n \to \{0,1\}^{2kw \log n}$, such that given $\mathbf{d}$, $I_{w+1}(\mathbf{c})$, and $H_w(\mathbf{c})$, we can recover that $I_w(\mathbf{c})$, for $w \in [k-1]$.*

Proof. The idea is to notice that given $I_{w+1}(\mathbf{c})$, the sequence $I_w(\mathbf{c})$ can be determined by the first 1 entry in $I_w(\mathbf{c})$ after each 1 entry in $I_{w+1}(\mathbf{c})$, $w \in [k-1]$. Specifically, let

$$(\pi_1^{w+1}, \pi_2^{w+1}, \ldots, \pi_{n'}^{w+1})$$

be the indices of the 1 entries in $I_{w+1}(\mathbf{c})$ such that $\pi_1^{w+1} < \pi_2^{w+1} < \ldots < \pi_{n'}^{w+1}$. Let

$$\tau_i^w = \min\{j : j > \pi_i^{w+1}, I_w(\mathbf{c})_j = 1 \text{ or } j = n+1\}$$

for $i \in [0, n']$, where $\pi_i^{w+1} = 0$ when $i = 0$. We have the following proposition.

Proposition 4.4.1. *The sequence $I_w(\mathbf{c})$ can be determined by $(\pi_1^{w+1}, \pi_2^{w+1}, \ldots, \pi_{n'}^{w+1})$ and $(\tau_0^w, \tau_1^w, \ldots, \tau_{n'}^w)$, for $w \in [k-1]$.*

Proof. Note that the 1 entries of $I_w(\mathbf{c})$ in the interval $(\pi_i^{w+1}, \pi_{i+1}^{w+1}]$ are spaced evenly with distance w in the interval $[\tau_i^w, \pi_{i+1}^{w+1}]$, for $i \in [0, n']$, where $\pi_{i+1}^{w+1} = n+1$ if $i = n'$. Otherwise there is an additional one entry in $I_{w+1}(\mathbf{c})$ in the interval $(\pi_i^{w+1}, \pi_{i+1}^{w+1}) \triangleq \{\pi_i^{w+1}+1, \dots, \pi_{i+1}^{w+1}-1\}$, which contradicts to the definition of $(\pi_1^{w+1}, \dots, \pi_{n'}^{w+1})$. □

From Proposition 4.4.1, it suffices to protect the indices $(\tau_0^w, \dots, \tau_{n'}^w)$, in order to recover $I_w(\mathbf{c})$. For $w \in [k-1]$, let

$$H_w(\mathbf{c}) = RS_{2kw}((\tau_0^w - \pi_0^{w+1}, \dots, \tau_{n'}^w - \pi_{n'}^{w+1})),$$

where $RS_{2kw}((\tau_0^w - \pi_0^{w+1}, \dots, \tau_{n'}^w - \pi_{n'}^{w+1}))$ is the redundancy of the Reed-Solomon code that corrects $2kw$ substitution errors in the sequence $(\tau_0^w - \pi_0^{w+1}, \dots, \tau_{n'}^w - \pi_{n'}^{w+1})$, with entries $\tau_i^w - \pi_i^{w+1}$, $i \in [0, n']$. The size of $H_w(\mathbf{c})$ is at most $4kw \log n$ bits.

In the following we present the decoding procedure that recovers $I_w(\mathbf{c})$, given $\mathbf{d} \in \mathcal{B}_k(\mathbf{c})$, $I_{w+1}(\mathbf{c})$, and $H_w(\mathbf{c})$, for any $w \in [k-1]$.

1. **Initialization:** Let $\mathbf{a} \in [n]^{n'+1}$ be a vector, where n' is known given $I_{w+1}(\mathbf{c})$.

2. **Step 1:** For each $i \in [0, n'-1]$, if there exist two numbers $p_i^{w+1} \in [\pi_i^{w+1} - k, \pi_i^{w+1}]$ and $p_{i+1}^{w+1} \in [\pi_{i+1}^{w+1} - k, \pi_{i+1}^{w+1}]$ such that $I_{w+1}(\mathbf{d})_{p_i^{w+1}} = I_{w+1}(\mathbf{d})_{p_{i+1}^{w+1}} = 1$ and $p_{i+1}^{w+1} - p_i^{w+1} = \pi_{i+1}^{w+1} - \pi_i^{w+1}$, let $j_i^w = \min_{j > p_i^{w+1}, I_w(\mathbf{d})_j = 1} j$ be the first 1 entry in $I_w(\mathbf{d})$ after $I_w(\mathbf{d})_{p_i^{w+1}}$, where $\mathbb{1}_w(\mathbf{d})_j = 1$ when $j = n - k + 1$. Let $a_i = j_i^w - p_i^w$. Else let $a_i = 0$.

3. **Step 2:** Apply the Reed-Solomon decoder on $\mathbf{a}$ to recover $(\tau_0^w - \pi_0^{w+1}, \dots, \tau_{n'}^w - \pi_{n'}^{w+1})$. Recover $(\tau_0^w, \dots, \tau_{n'}^w)$, and $I_w(\mathbf{c})$ according to Proposition 4.4.1.

4. **Step 3:** Output $I_w(\mathbf{c})$.

We now show that the above procedure decodes $I_w(\mathbf{c})$ correctly, $w \in [k-1]$. According to Lemma 4.4.1, The sequence $I_w(\mathbf{d})$ can be obtained from $I_w(\mathbf{c})$ after k deletions and at most $k(w-1)$ substitutions. Note that for each $i \in [0, n']$, we have that $a_i = \tau_i^w - \pi_i^{w+1}$, if no deletion or substitution occurs in the interval $[\pi_i^{w+1}, \pi_{i+1}^{w+1}]$ in $I_w(\mathbf{c})$, where $\pi_{i+1}^{w+1} = n$ if $i = n'$. Since a deletion or a substitution occurs in at most two adjacent intervals $[\pi_i^{w+1}, \pi_{i+1}^{w+1}]$ and $[\pi_{i+1}^{w+1}, \pi_{i+2}^{w+1}]$, k deletions and $k(w-1)$ substitutions cause at most $2kw$ symbol errors $a_i \neq \tau_i^w - \pi_i^{w+1}$ in $\mathbf{a}$. Hence the sequence $(\tau_0^w - \pi_0^{w+1}, \dots, \tau_{n'}^w - \pi_{n'}^{w+1})$, and thus $(\tau_0^w, \dots, \tau_{n'}^w)$ can be recovered given $H_w(\mathbf{c})$. Finally, according to Proposition 4.4.1, the sequence $I_w(\mathbf{c})$ can be recovered, $w \in [k-1]$.

The complexities for computing $H_w(\mathbf{c})$ and decoding $I_w(\mathbf{c})$ are dominated by encoding and decoding the Reed-Solomon code and are polynomial. □

Combining Lemma 4.4.2, Lemma 4.4.3, and Theorem 4.2.1, we now present the encoding and decoding as follows, which proves Theorem 4.1.2.

For any $\mathbf{c} \in \{0,1\}^n$, define the function

$$f_k^{sys}(\mathbf{c}) = (f(I_k(\mathbf{c})), H_1(\mathbf{c}), H_2(\mathbf{c}), \dots, H_{k-1}(\mathbf{c})). \tag{4.6}$$

We first show that $f_k^{sys}(\mathbf{c})$ is a k-deletion correcting labeling for $\mathbf{c}$.

Lemma 4.4.4. *For any $\mathbf{c} \in \{0,1\}^n$, $(\mathbf{c}, f_k^{sys}(\mathbf{c}))$ can be recovered from k deletions.*

Proof. For any length $n-k$ subsequence $\mathbf{d}$ of $\mathbf{c}$, according to Lemma 4.4.1, we have that $I_w(\mathbf{d}) \in \mathcal{B}_{k,k(w-1)}(I_w(\mathbf{c}))$ for $w \in [k]$. In particular, we have that $I_k(\mathbf{d}) \in \mathcal{B}_{k,k(k-1)}(I_k(\mathbf{c}))$. Since $I_k(\mathbf{d}) \in \mathcal{R}_{k,n-k}$, it follows from Lemma 4.4.2 that $f(I_k(\mathbf{c}) \neq f(I_k(\mathbf{c}'))$ for any $\mathbf{c}'$ satisfying $I_k(\mathbf{d}) \in \mathcal{B}_{k,k(k-1)}(I_w(\mathbf{c}'))$. Hence, the sequence $I_k(\mathbf{c})$ can be recovered, given $f(I_k(\mathbf{c}))$ and $\mathbf{d}$. According to Lemma 4.4.3, every sequence $I_w(\mathbf{c})$ can be recovered using $H_w(\mathbf{c})$, $I_{w+1}(\mathbf{c})$, and $\mathbf{d}$. Hence, after knowing $I_k(\mathbf{c})$, the sequence $\mathbf{c} = I_1(\mathbf{c})$ can be recovered by successively decoding $I_w(\mathbf{c})$, from $w = k-1$ to $w = 1$. □

The size of $f_k^{sys}(\mathbf{c})$ is $R = [(k^2+1)(2k^2+1) + 2k^2(k-1)]\log n + o(\log n)$, which is greater than $O(k \log n)$. By applying Theorem 4.2.1, there exists an integer $\alpha \in [2^{\log|\mathcal{B}_k(\mathbf{c})| + o(\log n)}] = [2^{2k\log n + o(\log n)}]$ such that $f_k^{sys}(\mathbf{c}) \not\equiv f_k^{sys}(\mathbf{c}') \bmod \alpha$ for any $\mathbf{c}' \in \mathcal{B}_k(\mathbf{c})$. Let

$$g_c(\mathbf{c}) = (f_k^{sys}(\mathbf{c}), \alpha).$$

Then $g_c(\mathbf{c})$ is a k-deletion correcting hash for $\mathbf{c}$ of size $N_1 = 4k\log n + o(\log n)$. Let

$$Hash_k(\mathbf{c}) = (g_c(\mathbf{c}), Rep_{k+1}(g_c(g_c(\mathbf{c})))),$$

where $Rep_{k+1}(g_c(g_c(\mathbf{c})))$ is the $k+1$ fold repetition of $g_c(g_c(\mathbf{c}))$, of length $N_2 = 4k\log N_1 + o(\log N_1) = 4k(k+1)\log\log n + o(\log n)$. The size of $Hash_k(\mathbf{c})$ is $N_1 + N_2 = 4k\log n + o(\log n)$. We now show that $Hash_k(\mathbf{c})$ is a k deletion correcting hash and thereby prove Theorem 4.1.2 For any length $n + N_1 + N_2 - k$ subsequence $\mathbf{z}$ of $(\mathbf{c}, Hash_k(\mathbf{c}))$, we have that $(z_{n+N_1+1}, \dots, z_{n+N_1+N_2-k})$ is a length $N_2 - k$ subsequence of $Rep_{k+1}(g_c(g_c(\mathbf{c})))$, which is a k-deletion correcting code. Therefore $g_c(g_c(\mathbf{c}))$ can be recovered. In addition, $(z_{n+1}, \dots, z_{n+N_1-k})$ is a length $N_1 - k$ subsequence of $g_c(\mathbf{c})$. Since $g_c(g_c(\mathbf{c}))$ is a k-deletion correcting hash of $g_c(\mathbf{c})$, the hash $g_c(\mathbf{c})$ can be recovered. Finally, note that $(z_1, \dots, z_{n-k})$ is a length $n - k$ subsequence of n, we can use $g_c(\mathbf{c})$ to recover $\mathbf{c}$. The decoding of $\mathbf{c}$ from $g_c(\mathbf{c})$ is done using brute force, over all sequences $\mathbf{c}'$ that satisfy $\mathbf{d} \in \mathcal{B}_k(\mathbf{c}')$. The computing of $g_c(\mathbf{c})$ is done by brute force, over sequences $\mathbf{c}' \in \mathcal{B}_k(\mathbf{c})$. Hence the encoding and decoding complexities are $O(n^{2k+1})$ and $O(n^{k+1})$ respectively.

4.5 Codes Capable of Correcting Bursts of Deletions

In this section, we consider the problem of constructing codes that can correct bursts of deletions. We consider two variants of this problem: (a) a single burst of at most k consecutive deletions occurs, and (b) at most k bursts of deletions each of length at most t_L occur. For (b), the deletions may or may not be consecutive. Similar to before, we will be interested in the setup where k and t_L are constants with respect to n. For (a), we show that our construction has $4\log n + o(\log n)$ bits of redundancy and for variant (b), our construction has $4k\log n + o(\log n)$ bits of redundancy. For (a) where the deletions occur consecutively, the construction is systematic.

Systematic codes capable of correcting a single burst of consecutive deletions

We begin by first introducing some notation. For a vector $\mathbf{x} \in \{0,1\}^m$, let $\mathcal{B}_k^{b_c}(\mathbf{x})$ denote the set of vectors that can result from deleting a burst of at most k consecutive symbols and then inserting symbols in a consecutive set of positions back into $\mathbf{x}$ to obtain a length m sequence. As an example if $\mathbf{x} = (0,1,1,0,1,0,1)$, then $\mathbf{y} = (1,0,1,1,1,0,1) \in \mathcal{B}_3^{b_c}(\mathbf{x})$, since $\mathbf{y}$ can be obtained from $\mathbf{x}$ by first deleting the first two symbols from $\mathbf{x}$ and then inserting the vector $(1,1)$ in position 3 of the resulting vector. It is straightforward so see that if $\mathbf{x}$ belongs to a code capable of correcting a burst of at most k consecutive symbols, then for any $\mathbf{y} \in \mathcal{B}_k^{b_c}(\mathbf{x})$, the vector $\mathbf{y}$ cannot be in the same code.

The next claim follows from elementary counting techniques.

Claim 4.5.1. *For any k and $\mathbf{u} \in \{0,1\}^m$,*

$$|\mathcal{B}_k^{b_c}(\mathbf{u})| \leq m^2 \cdot k2^{2k}.$$

In light of our syndrome compression technique, we need to describe the labeling function $f_k^{b_c}$. We define $f_k^{b_c}$ so that it is simply the result of computing the syndrome of a Varshamov-Tenengolts code $\frac{k(k+1)}{2}$ times. For notation convenience, suppose $i|n$ for $i \in [k]$, where $[k] = \{1,\dots,k\}$[2]. Then for a vector $\mathbf{u} \in \{0,1\}^n$, define $f_k^{b_c} : \{0,1\}^n \to [[\frac{n}{k}+1]]^k \times [[\frac{n}{k-1}+1]]^{k-1} \times \cdots \times [[n+1]]$ as

$$\begin{aligned} f_k^{b_c}(\mathbf{u}) = \Big(&\sum_{j=0}^{\frac{n}{k}-1} u_{k\cdot j+1} \bmod \frac{n}{k}+1, \sum_{j=0}^{\frac{n}{k}-1} u_{k\cdot j+2} \bmod \frac{n}{k}+1, \dots, \sum_{j=0}^{\frac{n}{k}-1} u_{k\cdot j+k} \bmod \frac{n}{k}+1 \\ &\sum_{j=0}^{\frac{n}{k-1}-1} u_{(k-1)\cdot j+1} \bmod \frac{n}{k-1}+1, \sum_{j=0}^{\frac{n}{k-1}-1} u_{(k-1)\cdot j+2} \bmod \frac{n}{k-1}+1, \dots, \\ &\sum_{j=0}^{\frac{n}{k-1}-1} u_{(k-1)\cdot j+k-1} \bmod \frac{n}{k-1}+1 \\ &\sum_{j=0}^{\frac{n}{k-2}-1} u_{(k-2)\cdot j+1} \bmod \frac{n}{k-2}+1, \sum_{j=0}^{\frac{n}{k-2}-1} u_{(k-2)\cdot j+2} \bmod \frac{n}{k-2}+1, \dots, \\ &\sum_{j=0}^{\frac{n}{k-2}-1} u_{(k-2)\cdot j+k-2} \bmod \frac{n}{k-2}+1 \\ &\vdots \\ &\sum_{j=1}^{n} u_j \bmod n+1 \Big) \in [[\frac{n}{k}+1]]^k \times [[\frac{n}{k-1}+1]]^{k-1} \times \cdots \times [[n+1]]. \end{aligned}$$

For convenience we will sometimes assume that the image of $f_k^{b_c}(\mathbf{u})$ is an integer from $\left[\left[\prod_{s=1}^{k}(\frac{n}{s}+1)^s\right]\right] \leq O(\frac{(n+k)^{k^2}}{k!})$. It is straightforward to show that $f_k^{b_c}$ satisfies the confusability property, but we include the following lemma for completeness.

Lemma 4.5.1. *Suppose* $\mathbf{u} \in \{0,1\}^n$. *Then for any* $\mathbf{y} \in \mathcal{B}_k^{b_c}(\mathbf{u})$,

$$f_k^{b_c}(\mathbf{u}) \neq f_k^{b_c}(\mathbf{y}).$$

Proof. To prove the result, assume that $\mathbf{z}$ is the result of a burst of deletions of length at most k occurring to $\mathbf{u}$ and we are given $f_k^{b_c}(\mathbf{u})$. We will show that it is possible to uniquely recover $\mathbf{u}$ from $\mathbf{z}$ given $f_k^{b_c}(\mathbf{u})$, which is equivalent to showing that for any $\mathbf{y} \in \mathcal{B}_k^{b_c}(\mathbf{u})$, $f_k^{b_c}(\mathbf{u}) \neq f_k^{b_c}(\mathbf{y})$.

Suppose $f_k^{b_c}(\mathbf{u}) = (a_{k,1},\dots,a_{k,k},a_{1,k-1},\dots,a_{k-1,k-1},\dots,a_1)$ and that $|\boldsymbol{z}| = n-s$ so that $\mathbf{z}$ is the result of a burst of $s \leq k$ consecutive deletions occurring to $\mathbf{u}$. Consider the sequences:

$$\mathbf{z}^{(1)} = (z_1, z_{1+s}, z_{1+2s}, \dots, z_{n-s+1}),$$

[2]We can replace $\frac{n}{i}$ with $\lceil \frac{n}{i} \rceil$ if $i \nmid n$.

$$\mathbf{z}^{(2)} = (z_2, z_{2+s}, z_{2+2s}, \ldots, z_{n-s+2}),$$
$$\vdots$$
$$\mathbf{z}^{(s)} = (z_s, z_{2s}, z_{3s}, \ldots, z_n).$$

Also, let

$$\mathbf{u}^{(1)} = (u_1, u_{1+s}, u_{1+2s}, \ldots, u_{n-s+1}),$$
$$\mathbf{u}^{(2)} = (u_2, u_{2+s}, u_{2+2s}, \ldots, u_{n-s+2}),$$
$$\vdots$$
$$\mathbf{u}^{(s)} = (u_s, u_{2s}, u_{3s}, \ldots, u_n).$$

Since for $i \in [s]$, $\mathbf{z}^{(i)}$ is the result of a single deletion occurring to $\mathbf{u}^{(i)}$, it is possible to recover $\mathbf{u}^{(i)}$ given $\mathbf{z}^{(i)}$ and $a_{s,1}, a_{s,2}, \ldots, a_{s,s}$ since

$$\left\{\mathbf{u}^{(i)} = (u_1^{(i)}, \ldots, u_{\frac{n}{s}}^{(i)}) \in \{0,1\}^{\frac{n}{s}} : \sum_{j=1}^{\frac{n}{s}} u_j^{(i)} \equiv a_{s,i} \bmod \frac{n}{s} + 1\right\}$$

is a code capable of correcting a single deletion. □

From Lemma 4.5.1 the mapping $f_k^{b_c}$ satisfies the confusability property. Furthermore, $f_k^{b_c}$ satisfies the redundancy property since $\log\left(\prod_{s=1}^{k}(\frac{n}{s}+1)^s\right) \leq O\left(k^2 \log(n+k)\right)$, and k is assumed to be a constant. Therefore, from Theorem 4.2.1, for any $\mathbf{u} \in \{0,1\}^n$ there exists an integer a such that $a \leq 2^{\log|\mathcal{B}_k^{b_c}(\mathbf{u})| + o(\log n)}$, and for any $\mathbf{y} \in \mathcal{B}_k^{b_c}(\mathbf{u})$, $f_k^{b_c}(\mathbf{u}) \not\equiv f_k^{b_c}(\mathbf{y}) \bmod a$.

We define our code $C^{b_c}(N,k)$ with $N = n + 2\log|\mathcal{B}_k^{b_c}(\mathbf{x})| + o(\log n)$ as follows:

$$C^{b_c}(N,k) = \left\{\mathbf{x} = \left(\mathbf{u}, 1, \mathbf{0}^k, \mathbf{1}^k, 0, a,\ f_k^{b_c}(\mathbf{u}) \bmod a\right) : \mathbf{u} \in \{0,1\}^n\right\}. \quad (4.7)$$

We now prove the following theorem and thus prove Theorem 4.1.3. In the statement below, $\mathbf{u}$ is the information portion of the sequence (the non-redundancy part) from (4.7).

Theorem 4.5.1. *Let* $\mathbf{z}$ *be the result of a consecutive burst of length at most* k *occurring to* $\mathbf{x} \in C^{b_c}(N, k)$*. Then, we can uniquely determine* $\mathbf{x}$ *from* $\mathbf{z}$*.*

Proof. To prove the result, we show how to recover $\mathbf{u}$ from $\mathbf{z}$. In order to recover $\mathbf{u}$ from $\mathbf{z}$, we show that it is possible to separate $\mathbf{z}$ into two parts: $\mathbf{z}_1$ and $\mathbf{z}_2$ where either a) $\mathbf{z}_1$ is the result of a burst of deletions of length at most k occurring to $\mathbf{u}$ or b) $\mathbf{z}_2$ is the result of a burst of deletions of length at most k occurring to $\mathbf{r} = (1,\ 0^k,\ 1^k,\ 0,\ a,\ f_k^{b_c}(\mathbf{u}) \bmod a)$. Note that if $\mathbf{z}_1 \neq \mathbf{u}$, then (due to the length of the burst) $\mathbf{z}_2 = \mathbf{r}$ and the fact that we can recover $\mathbf{u}$ from $\mathbf{z}_1$ provided $\mathbf{r}$ follows immediately from Theorem 4.2.1. If b) holds and $\mathbf{z}_2 \neq \mathbf{r}$, then by similar logic, $\mathbf{u} = \mathbf{z}_1$. Note that the fact that $\mathbf{z}_1 \neq \mathbf{u}$ can be determined immediately by the length of $\mathbf{z}_1$ (due to the deletions) and similarly we can easily detect when $\mathbf{z}_2 \neq \mathbf{r}$ by considering the length of $\mathbf{z}_2$. Therefore, in the remainder of the proof we show how to recover $\mathbf{z}_1, \mathbf{z}_2$ from $\mathbf{z}$ assuming a burst of $s \leq k$ deletions have occurred to $\mathbf{x}$ resulting in $\mathbf{z}$.

In order to separate $\mathbf{z}$ into $\mathbf{z}_1$ and $\mathbf{z}_2$, we make use of the marker sequence $1,\ 0^k,\ 1^k,\ 0,$ which is embedded into every codeword in our code according to (4.7). Let $|\mathbf{z}| = n - s$. If

$$(z_{n+1}, z_{n+2}, \ldots, z_{n+2k-s+1}) = (0^{k-s+1}, 1^k), \tag{4.8}$$

then, it is straightforward to observe that $\mathbf{z}_2 = \mathbf{r}$ where z_2 is equal to the last $N - n$ bits of $\mathbf{z}$. We set $\mathbf{z}_1$ to be equal to the first $n - s$ bits of $\mathbf{z}$ so that by the previous discussion we can recover $\mathbf{u}$ from $\mathbf{z}$.

Next, suppose that

$$(z_{n+1}, z_{n+2}, \ldots, z_{n+k+1}) = (1, 0^k). \tag{4.9}$$

In this case the burst of length k could not have started in any of the positions from the set $[n] = \{1, 2, \ldots, n\}$, which implies $\mathbf{u}$ is equal to the first n bits of $\mathbf{z}$.

The only case left to consider is where the deletion begins in marker sequence $1,\ 0^k,\ 1^k,\ 0$. First note that if the deletion occurs in the marker sequence then (4.8) can hold only if the deletion begins in position $n + 1$ in $\mathbf{x}$. In this case, it is straightforward to verify that the decoding described for this will still generate $\mathbf{u}$ since $\mathbf{r}$ is still equal to the last $N - n$ bits of $\mathbf{z}$. If the deletion begins in one of the positions $\{n + 2, n + 3, \ldots, n + k + 1\}$, then

$$(z_{n+1}, z_{n+2}, \ldots, z_{n+1+k}) = (1, 0^j, 1^{k-j}),$$

so that neither (4.8) or (4.9) can hold. If the deletion begins in the marker sequence after position $n+k+1$ in $\mathbf{x}$, then (4.9) holds and the decoding is correct in this case as well. $\square$

Codes correcting bursts of deletions

Next, we consider a more generalized type of burst error pattern. In this section, we want to correct k bursts each occurring within a window of length at most t_L where the deletions in each burst need not occur consecutively. For shorthand, we refer to these codes as (k, t_L)-burst codes. The main result here will be to show that for the case where k, t_L are constants, there exists (k, t_L)-burst codes with redundancy $4k(1+\epsilon)\log n$ for k, n large enough.

We begin by first introducing some notation, and then we proceed to our code construction. We say that $\mathbf{z} \in \{0,1\}^{n-|J|}$ is the result of k bursts each occurring within a window of length at most t_L occurring to $\mathbf{x} \in \{0,1\}^n$ if there exists sets $J, J_b \subseteq [n]$, with $|J| \le k \cdot t_L$, $|J_b| = k$ such that the following holds:

1. $\mathbf{z}$ can be obtained by deleting symbols from $\mathbf{x}$ in positions J.
2. For any $j \in J$, there exists an $i \in J_b$ where $|j - i| < t_L$.

We illustrate these notations in the following example.

Example 4.5.1. *Suppose* $\mathbf{x} = (0,1,1,\cancel{1},0,\cancel{1},0,0,0,1,1,\cancel{0},0) \in \{0,1\}^{13}$ *is in a* $(2,3)$*-burst code. Let*

$$\mathbf{z} = (0,1,1,0,0,0,0,1,1,0)^{10}.$$

Then, we can claim that $\mathbf{z}$ *is the result of* 2 *bursts of deletions of length at most* 3 *since we can write* $J = \{4,6,12\}$ *and* $J_b = \{4,12\}$ *with* $t_L = 3$*. It follows that given* $\mathbf{z}$*, it is possible to uniquely recover* $\mathbf{x}$ *provided* $\mathbf{x}$ *is in a* $(2,3)$*-burst code.*

For a vector $\mathbf{x} \in \{0,1\}^m$, let $B_{k,t_L}(\mathbf{x})$ be the set of vectors possible given that k bursts each occurring within a window of length at most t_L occur to $\mathbf{x}$. Then, define $\mathcal{B}^b_{k,t_L}(\mathbf{x}) \subseteq \{0,1\}^m$ so that

$$\mathcal{B}^b_{k,t_L}(\mathbf{x}) = \{\mathbf{y} \in \{0,1\}^m : B_{k,t_L}(\mathbf{x}) \cap B_{k,t_L}(\mathbf{y}) \neq \emptyset, \mathbf{y} \neq \mathbf{x}\}.$$

Clearly, if $\mathbf{x}$ is in a (k, t_L)-burst code, then $\mathbf{y}$ cannot be in the same code for any $\mathbf{y} \in \mathcal{B}^b_{k,t_L}(\mathbf{x})$. The following claim follows from straightforward counting arguments.

Claim 4.5.2. *For integers k, t_L, m, and any $\mathbf{u} \in \{0,1\}^m$,*

$$|\mathcal{B}^b_{k,t_L}(\mathbf{u})| \le m^{2k} \cdot \left((t_L+1)^k 2^{k\cdot t_L}\right)^2.$$

In order to apply the syndrome compression technique, we need to specify the labeling and also to show that the redundancy and confusability properties hold. For this setup, we will use the same systematic labeling used to correct multiple deletions that was introduced in Sec. 4.4. More specifically, we will use the labeling g defined in (4.6). It follows immediately from our definitions and Lemma 4.4.4 that if $\mathbf{u}, \mathbf{y} \in \{0,1\}^n$ and $\mathbf{y} \in \mathcal{B}^b_{k,t_L}(\mathbf{u}) \backslash \{\mathbf{u}\}$, then

$$f^{sys}_k(\mathbf{u}) \neq f^{sys}_k(\mathbf{y}),$$

so that the confusability property holds. The redundancy property also follows immediately from the definition of g since k, t_L are constants. Thus, to construct (k, t_L)-burst codes, we can apply the same syndrome compression procedure as described in Sec. 4.3 and Sec. 4.4, except that we will search for an $a \in [[n^{2k} \cdot \left((t_L+1)^k 2^{k\cdot t_L}\right)^2]]$ such that $f^{sys}_k(\mathbf{u}) \not\equiv f^{sys}_k(\mathbf{y}) \bmod a$ for any $\mathbf{y} \in \mathcal{B}^b_{k,t_L}(\mathbf{u})$. Since $\log a \le 2k \log n + o(\log n)$ for n large enough, the resulting construction is systematic and has redundancy $4k \log n + o(\log n)$ for k, n large enough. Hence, we have Theorem 4.1.4.

4.6 q-ary Codes Correcting k Deletions for Small q

In this section we present k deletion correcting codes for q-ary alphabets where q is less than the information length n. In particular, we consider the following two cases: (1) $q \le \log n$. (2) $\log n < q \le n$. The redundancy of the resulting q-ary k deletion codes for case (1) and (2) are $4k \log n + o(\log n)$ and $2k(1+\epsilon)(2\log n + \log q) + o(\log n)$ bits, respectively, where n is the information length. Before describing the code constructions, let us introduce a few notations for this section. For a sequence $\mathbf{u} \in [[q]]^n$ of length n over the alphabet $\{0, 1, \ldots, q-1\}$, let $\mathcal{B}^q_k(\mathbf{u})$ be its deletion ball with radius k, consisting of all length m sequences obtained by deleting k q-ary symbols and inserting k q-ary symbols in $\mathbf{u}$. The following result comes from a simple counting argument.

Claim 4.6.1. *For any k and $\mathbf{u} \in [[q]]^n$,*

$$|\mathcal{B}^q_k(\mathbf{u})| \le n^{2k} q^k.$$

Define a binary matrix representation U for $\mathbf{u} \in [[q]]^n$ as

$$U = \begin{bmatrix} u_{1,1} & u_{1,2} & \dots & u_{1,m} \\ \vdots & \ddots & \vdots & \\ u_{\lceil \log q \rceil,1} & u_{\lceil \log q \rceil,2} & \dots & u_{\lceil \log q \rceil,m} \end{bmatrix} \in \{0,1\}^{\lceil \log q \rceil \times m}, \qquad (4.10)$$

where the i-th symbol of $\mathbf{u}$ is given by the i-th column of U for $i \in [n]$. Let U_i^r, $i \in [\lceil \log q \rceil]$ and U_j^c, $j \in [n]$ be the i-th row and j-th column of U respectively. Then the deletion of the j-th symbol of $\mathbf{u}$ corresponds to the deletion of the column U_j^c in the matrix U.

Case $(1) : q \leq \log n$
In the following, we adapt the syndrome compression technique to correct deletions for the case where $q \leq \log n$. The basic idea, which will be described in more details that follow, is to interpret our non-binary sequences as a set of $\log q$ sequences over the binary alphabet as illustrated in (4.10). We will then use a compound labeling which is defined using the systematic labeling from Sec. 4.4 on each of these binary sequences to form a code that can correct k deletions.

We begin by describing the labeling. According to Lemma 4.4.4, there exists a systematic labeling function

$$f_k^{sys}(\mathbf{u}) : \{0,1\}^m \to \{0,1\}^{R=[(k^2+1)(2k^2+1)+2k^2(k-1)]\log n + o(\log n)}$$

such that $f_k^{sys}(\mathbf{u}) \neq f_k^{sys}(\mathbf{y})$ for any $\mathbf{u}, \mathbf{y} \in \{0,1\}^n$ and $\mathbf{y} \in \mathcal{B}_k(\mathbf{u}) \backslash \{\mathbf{u}\}$. Define the labeling function

$$f_k^q(\mathbf{u}) = \left(f_k^{sys}(U_1^r), \dots, f_k^{sys}(U_{\log q}^r) \right) \in \{0,1\}^{\lceil \log q \rceil R}.$$

Then, from Lemma 4.4.4, we have that

$$f_k^q(\mathbf{u}) \neq f_k^q(\mathbf{y})$$

for $\mathbf{u}, \mathbf{y} \in [[q]]^n$ and $\mathbf{y} \in \mathcal{B}_k^q(\mathbf{u}) \backslash \{\mathbf{u}\}$. Hence the labeling function $f_k^q(\mathbf{u})$ satisfies the confusability. The size of $f_k^q(\mathbf{u})$ is $R_q = O([(k^2+1)(2k^2+1) + 2k^2(k-1)] \log q \log n)$ bits, which implies that the labeling $f_k^q(\mathbf{u})$ does not have the redundancy property, so that we cannot immediately apply our syndrome compression technique like was done in Sec. 4.3. To resolve this issue, in the next lemma the idea is to basically use the syndrome compression technique twice on $f_k^q(\mathbf{u})$ in order to generate a new labeling $f_k^{q_1}$.

Lemma 4.6.1. *There exists a labeling $f_k^{q_1}$ that satisfies the redundancy property and $f_k^{q_1}$ is such that for any $\mathbf{u}, \mathbf{y} \in [[q]]^n$ and $\mathbf{y} \in \mathcal{B}_k^q(\mathbf{u}) \backslash \{\mathbf{y}\}$, we have*

$$f_k^{q_1}(\mathbf{u}) \neq f_k^{q_1}(\mathbf{y}).$$

Proof. Define the set

$$\mathcal{D}(\mathbf{u}) = \left\{ j : j > 0, j | (f_k^q(\mathbf{u}) - f_k^q(\mathbf{y})) \text{ for some } \mathbf{y} \in [[q]] \cap \mathcal{B}_k^q(\mathbf{u}) \right\}$$

for $\mathbf{u} \in [[q]]^n$. According to Lemma 4.2.1, we have that

$$|\mathcal{D}(\mathbf{u})| \leq 2^{\log |\mathcal{B}_k^q(\mathbf{u})| + \frac{1.6}{\log e} \cdot \frac{Rq}{\ln \frac{Rq}{\log e}}} \leq 2^{2k \log n + k \log q + \frac{1.6}{\log e} \cdot \frac{Rq}{\ln \frac{Rq}{\log e}}} = 2^{O(poly(k) \log n)}.$$

Therefore, there exists an integer $\alpha_{\mathbf{u}} \leq 2^{O(poly(k) \log n)}$ such that $f_k^q(\mathbf{u}) \not\equiv f_k^q(\mathbf{y}) \bmod \alpha_{\mathbf{u}}$ and thus $(\alpha_{\mathbf{u}}, f_k^q(\mathbf{u}) \bmod \alpha_{\mathbf{u}}) \neq (\alpha_{\mathbf{y}}, f_k^q(\mathbf{y}) \bmod \alpha_{\mathbf{y}})$ for $\mathbf{u}, \mathbf{y} \in [[q]]^n$ and $\mathbf{y} \in \mathcal{B}_k^q(\mathbf{u}) \backslash \{\mathbf{u}\}$. Let

$$f_k^{q_1}(\mathbf{u}) \triangleq (\alpha_{\mathbf{u}}, f_k^q(\mathbf{u}) \bmod \alpha_{\mathbf{u}}). \tag{4.11}$$

The fact that $f_k^{q_1}(\mathbf{u}) \neq f_k^{q_1}(\mathbf{y})$ follows from the previous discussion. From (4.11), the length of $f_k^{q_1}(\mathbf{u})$ is $O(poly(k) \log m)$. Hence $f_k^{q_1}$ satisfies the redundancy property, and this completes the proof. □

We are now ready to present our code construction in terms of the encoding process. Let $Hash_k$ be the encoder from Theorem 4.1.2 which takes as input any binary information sequence and outputs a deletion correcting hash.

1. Let $\mathbf{u} \in [[q]]^n$.

2. Suppose $a \in [[2^R]]$ is such that $f_k^{q_1}(\mathbf{u}) \not\equiv f_k^{q_1}(\mathbf{y}) \bmod a$ for any $\mathbf{y} \in \mathcal{B}_k^q(\mathbf{u}) \backslash \{\mathbf{y}\}$ where $R = 2k \log n + o(\log n)$.

3. Then,

$$\mathbf{x} = \Big(\mathbf{u}, a, f_k^{q_1}(\mathbf{u}) \bmod a, Hash_k^q\big(a, f_k^{q_1}(\mathbf{u}) \bmod a\big) \Big) \in [[q]]^n.$$

In the resulting codeword $\mathbf{x}$ above, we assume that a and $f_k^{q_1}(\mathbf{u}) \bmod a$ are represented using q-ary symbols and the vector $Hash_k^q\big(a, f_k^{q_1}(\mathbf{u}) \bmod a\big)$ takes the hash $Hash_k$ for each row in the matrix representation of $a, f_k^{q_1}(\mathbf{u}) \bmod a$.

Since $f_k^{q_1}$ satisfies the redundancy property, it follows from Sec. 4.2 that $(a, f_k^{q_1}(\mathbf{u}) \bmod a)$ can be described using at most $4k \log n + o(\log n)$ bits. This implies that $Hash_k^q\big(a, f_k^{q_1}(\mathbf{u}) \bmod a\big)$ can be represented with at most $4k \log\log n + o(\log\log n)$ bits and so the redundancy of the encoding is at most $4k \log n + o(\log n)$ bits of redundancy. Together with the following theorem, we prove Theorem 4.1.5 for cases when $q \leq \log n$.

Theorem 4.6.1. *Let* $\mathbf{z}$ *be the result of at most* k *deletions occurring to* $\mathbf{x}$*. Then, we can uniquely recover* $\mathbf{x}$ *from* $\mathbf{z}$*.*

Proof. Let $\mathbf{z}$ be a length $N - k$ subsequence of $\mathbf{x}$, where N is the length of $\mathbf{x}$. Then $(z_{n+1}, \ldots, z_{N-k})$ is a length $N - k - n$ subsequence of $(x_{n+1}, \ldots, x_N)$. Since $(x_{k+1}, \ldots, x_n) = a, f_k^{q_1}(\mathbf{u}) \bmod a, Hash_k^q\big(a, f_k^{q_1}(\mathbf{u}) \bmod a\big)$ is a codeword from a k-deletion correcting code, we can recover $a, f_k^{q_1}(\mathbf{u}) \bmod a$ from $(z_{n+1}, \ldots, z_{N-k})$. We have that $f_k^{q_1}(\mathbf{y}) \not\equiv f_k^{q_1}(T_1^q(\mathbf{u}, s)) \bmod a$ for $\mathbf{y} \in \mathcal{B}_k^q(\mathbf{u}) \backslash \{\mathbf{u}\}$. Therefore, the sequence $\mathbf{u}$ can be recovered. □

Case (2) : $\log n < q \leq n$

We now present a k deletion code for the case $\log n < q < n$. Two key ideas are involved. The first is to narrow down the ranges of deletion locations to intervals of length $O(poly(k) \log n)$, thus recovering most of the symbols in the sequence. To further recover the rest of the symbols, the second idea decomposes a q-ary representation of a sequence down to its symbol histogram information, which counts the frequency of the symbols, and its permutation information, which records the index of each symbol. The second idea can be regarded as a generalization of the construction of q-ary single deletion correcting codes in [93], where the symbol histogram and ascending/descending order are considered separately.

To achieve the first part, we generate binary sequences that satisfy a period constraint, which was also defined in [18]. A sequence $\mathbf{u} \in \{0, 1\}^n$ has period p if $\mathbf{u}_i = \mathbf{u}_{i+p}$ for $i \in [[n-p]]$. Let $L(\mathbf{u}, p)$ be the length of the longest subsequence of consecutive bits in $\mathbf{u}$ that has period p. Denote

$$\mathcal{L}(n, k) = \Big\{\mathbf{u} : L(\mathbf{u}, p) \leq 2\log n + k + 1, \forall p \in [[k]]\Big\}$$

to be the set of sequences whose subsequences of any period $p \in [[k]]$ has length not greater than $2 \log n + k + 1$.

Lemma 4.6.2. *Let* U_ℓ *be a random string uniformly distributed over* $\{0, 1\}^\ell$ *where* $\ell = 2\log n + k + 1$*. Let* $g_1(U_\ell) \in \{0, 1\}^n$ *be the sequence obtained by repeating* U_ℓ

and then taking the first n bits. Then for any sequence $\mathbf{u} \in \{0,1\}^n$*, the bitwise XOR* $g_1(U_\ell) + \mathbf{u} \in \mathcal{L}(n,k)$ *with probability at least* $1 - 1/n$*.*

Proof. Let $\mathbf{y} = g_1(U_\ell) + \boldsymbol{c}$ be the bitwise XOR sequence. The probability of the event $L((\mathbf{y}_i, \mathbf{y}_{i+1}, \ldots, \mathbf{y}_{i+2\log n+k}), p) = 2\log n + k + 1$ is given by $2^p/2^{2\log n+k+1} = 2^p/(2^{k+1}n^2)$ for any $i \in [[n - 2\log n - k]]$ and $p \in [[k]]$. By the union bound, the probability that $L((\mathbf{y}_i, \mathbf{y}_{i+1}, \ldots, \mathbf{y}_{i+2\log n+k}), p) = 2\log n + k + 1$ for some index $i \in [[n - 2\log n - k]]$ and period $p \in [[k]]$ is upper bounded by

$$\sum_{i\in[[n-2\log n-k]],p\in[[k]]} 2^p/(2^{k+1}n^2) \leq 1/n.$$

Therefore, the probability that $\mathbf{y}$ does not contain length $2\log n + k + 1$ subsequence of period $p \in [[k]]$ is at least $1 - 1/n$ and the proof is done. □

For a sequence $\mathbf{u} \in \{0,1\}^n$ and integers $\{\delta_1, \ldots, \delta_k\}$, where $1 \leq \delta_1 < \delta_2 < \ldots < \delta_k \leq n$, let $\mathbf{u}(\delta_1, \ldots, \delta_k)$ denote the length $n-k$ subsequence obtained by deleting bits $\mathbf{u}_{\delta_i}, i \in [[k]]$. The next lemma shows that given $\mathbf{u} \in \mathcal{L}(n,k)$ and $\mathbf{u}(\delta_1, \ldots, \delta_k)$, it is possible to narrow down the range of $\delta_i, i \in [[k]]$.

Lemma 4.6.3. *If a sequence* $\mathbf{u} \in \mathcal{L}(n,k)$*, then given* $\mathbf{u}$ *and* $\mathbf{u}(\delta_1, \ldots, \delta_k)$*, we can find at most* $k+2$ *disjoint intervals* $[a_i, b_i] \subset [[n]]$ *for* $i \in [[k+2]]$*, with length* $O(poly(k)\log n)$ *each, such that* $\{\delta_1, \ldots, \delta_k\} \subset \cup_{i\in[1,k+2]}[a_i, b_i]$*. In addition, for any* $j \in [[n]]\backslash(\cup_{i\in[[k+2]]}[a_i, b_i])$*, the number of deletions* $N_j = |[[j-1]] \cap \{\delta_1, \ldots, \delta_k\}|$ *that occur in interval* $[[j-1]]$ *can be determined.*

Proof. Let $\boldsymbol{z} = \mathbf{u}(\delta_1, \ldots, \delta_k)$. The algorithm for finding $[a_i, b_i], i \in [[k+2]]$ is as follows.

1. **Initialization:** Set all indices $[[n]]$ unmarked. Let $j = 1$. Go to Step 1.

2. **Step 1:** Find the the largest positive integer L such that

$$(\mathbf{z}_j, \mathbf{z}_{j+1}, \ldots, \mathbf{z}_{j+L-1-p}) = (\mathbf{z}_{j+p}, \mathbf{z}_{j+p+1}, \ldots, \mathbf{z}_{j+L-1}) \tag{4.12}$$

   for some $p \in [[k]]$, so that a common length $L - p$ substring exists in both $\mathbf{z}$ and $\mathbf{u}$. If $L \geq (2k+1)(2\log n + k + 2) + k$, mark the numbers w in interval $[j + k(2\log n + k + 2) + k, j + L - k(2\log n + k + 2) - 1]$ and let $N_w = p$ for $w \in [j+k(2\log n+k+2)+k, j+L-k(2\log n+k+2)-1]$. Let $j = j+L-p$ and repeat. Else go to Step 2.

3. **Step 2:** If an unmarked interval has length greater than $(2k+1)(2\log n+k+2)+k$, then split the interval into intervals of length $(2k+1)(2\log n+k+2)+k$ except that the last interval has length not greater than $(2k+1)(2\log n+k+2)+k$. Output all unmarked intervals and N_w for w in marked intervals.

We first show by contradiction that if $L \geq (2k+1)(2\log n+k+2)+k$, then no deletions happen in the marked interval $[j+k(2\log n+k+2)+k, j+L-k(2\log n+k+2)-1]$ in Step 1, i.e., $\{\delta_1,\dots,\delta_k\}\cap[j+k(2\log n+k+2)+k, j+L-k(2\log n+k+2)-1]=\emptyset$. Suppose on the contrary, there exists some $\ell \in [[k]]$ such that $\delta_\ell \in [j+k(2\log n+k+2)+k, j+L-k(2\log n+k+2)-1]$. Then, using a Pigeonhole argument, since $\delta_\ell - j - k \geq k(2\log n+k+2)$, there exist δ_{ℓ_1} for some $\ell_1 \in [0,\ell-1]$, such that $\delta_{\ell_1+1}-\max\{\delta_{\ell_1}, j+k\} \geq 2\log n+k+2$. With a slight abuse of notation, when $\ell_1=0$, we assume $\delta_{\ell_1}=0$ for convenience.

By definition of δ_{ℓ_1} and δ_{ℓ_1+1}, we have that

$$(\mathbf{z}_{\max\{\delta_{\ell_1},j+k\}+1-\ell_1},\dots,\mathbf{z}_{\delta_{\ell_1+1}-1-\ell_1}) = (\mathbf{z}_{\max\{\delta_{\ell_1},j+k\}+1},\dots,\mathbf{z}_{\delta_{\ell_1+1}-1}). \tag{4.13}$$

We now show that the p in Eq. (8.8) equals ℓ_1. If $p>\ell_1$, then according to Eq. (4.13) and Eq. (8.8), we have

$$\begin{aligned}(\mathbf{u}_{\max\{\delta_{\ell_1},j+k\}+1+p-\ell_1},\dots,\mathbf{u}_{\delta_{\ell_1+1}-1-\ell_1+p}) =&(\mathbf{z}_{\max\{\delta_{\ell_1},j+k\}+1-\ell_1},\dots,\mathbf{z}_{\delta_{\ell_1+1}-1-\ell_1})\\ =&(\mathbf{u}_{\max\{\delta_{\ell_1},j+k\}+1},\dots,\mathbf{u}_{\delta_{\ell_1+1}-1}).\end{aligned}$$

This implies that

$$\begin{aligned}L((\mathbf{u}_{\max\{\delta_{\ell_1},j+k\}+1},\dots,\mathbf{u}_{\delta_{\ell_1+1}-1-\ell_1+p}),p-\ell_1) =&\delta_{\ell_1+1}-\max\{\delta_{\ell_1},j+k\}+p-\ell_1-1\\ \geq&2\log n+k+2,\end{aligned}$$

which is a contradiction to the fact that $\mathbf{u}\in\mathcal{L}(n,k)$. Similarly, if $p<\ell_1$, we have that

$$\begin{aligned}(\mathbf{u}_{\max\{\delta_{\ell_1},j+k\}-p+\ell_1+1},\dots,\mathbf{u}_{\delta_{\ell_1+1}-1}) =&(\mathbf{z}_{\max\{\delta_{\ell_1},j+k\}-p+1},\dots,\mathbf{z}_{\delta_{\ell_1+1}-1-\ell_1})\\ =&(\mathbf{u}_{\max\{\delta_{\ell_1},j+k\}+1},\dots,\mathbf{u}_{\delta_{\ell_1+1}-1-\ell_1+p}),\end{aligned}$$

and thus that

$$\begin{aligned}L((\mathbf{u}_{\max\{\delta_{\ell_1},j+k\}+1},\dots,\mathbf{u}_{\delta_{\ell_1+1}-1-p+\ell_1}),\ell_1-p) =&\delta_{\ell_1+1}-\max\{\delta_{\ell_1},j+k\}+\ell_1-p-1\\ \geq&2\log n+k+2,\end{aligned}$$

which contradicts to the fact that $\mathbf{u} \in \mathcal{L}(n,k)$. Therefore, we have that $p = \ell_1$.

On the other hand, since $j + L - 1 - \delta_\ell \geq k(2\log n + k + 2)$, there exist δ_{ℓ_2} for some $\ell_2 \in [\ell, k]$, such that $\min\{\delta_{\ell_2+1}, j + L - 1\} - \delta_{\ell_2} \geq 2\log n + k + 2$, where for notation convenience, it is assumed that $\delta_{\ell_2+1} = n + 1$ when $\ell_2 = k$. The same argument above can be used to show that $p = \ell_2$, which leads to a contradiction since $p = \ell_1$ and $\ell_1 < \ell_2$. Therefore, we conclude that the marked intervals do not contain deletions.

Next we show that $N_w = p = |[1, j+k(2\log n+k+2)+k-1] \cap \{\delta_1, \ldots, \delta_k\}|$ for the marked interval $w \in [j+k(2\log n+k+2)+k, j+L-k(2\log n+k+2)-1]$. Let $k' = |[1, j+k(2\log n+k+2)+k-1] \cap \{\delta_1, \ldots, \delta_k\}|$. Since $L \geq (2k+1)(2\log n+k+2)$ and no deletions occur in the interval $[j+k(2\log n+k+2)+k, j+L-k(2\log n+k+2)-1]$, we have that $\min\{\delta_{k'+1}, j+L-1\} - \max\{\delta_{k'}, j+k\} \geq 2\log n+k+2$, where $\delta_{k'+1} = n$ if $k' = k$ and $\delta_{k'} = 0$ if $k' = 0$. By the same arguments proving $p = k_1$, we have that $p = k' = N_w$ for $w \in [j + k(2\log n + k + 2) + k, j + L - k(2\log n + k + 2) - 1]$.

Finally, we show that there are at most $k + 2$ unmarked intervals, each with length at most $(2k + 1)(2\log n + k + 2) + k$. Note that any unmarked index w' satisfies one of the following:

1. $w' \in [[(2k + 1)(2\log n + k + 2) + k]]$,
2. $\{\delta_1, \ldots, \delta_k\} \cap [w' - \lfloor[(2k+1)(2\log n+k+2)+k]/2\rfloor, w' + \lfloor[(2k+1)(2\log n+k+2)+k]/2\rfloor] \neq \emptyset$,
3. $w' \in [n - (2k + 1)(2\log n + k + 2) - k + 1, n]$.

Otherwise if there exists some w' such that $w' \notin [[(2k + 1)(2\log n + k + 2) + k]] \cup [n-(2k+1)(2\log n+k+2)-k+1, n]$ and $\{\delta_1, \ldots, \delta_k\} \cap [w' - \lfloor[(2k+1)(2\log n+k+2)+k]/2\rfloor, w' + \lfloor[(2k+1)(2\log n+k+2)+k]/2\rfloor] = \emptyset$, then we have that $\mathbf{u}_w = z_{w-p}$, where $p = |\{\delta_1, \ldots, \delta_k\} \cap [[w' - w' - \lfloor[(2k + 1)(2\log n + k + 2) + k]/2\rfloor - 1]]|$, for every $w \in [w' - \lfloor[(2k + 1)(2\log n + k + 2) + k]/2\rfloor, w' + \lfloor[(2k + 1)(2\log n + k + 2) + k]/2\rfloor]$. Then the interval $[w' - \log n - 1, w' + \log n + k + 1]$ is marked, which contradicts to the fact that w' is unmarked. Therefore, the unmarked intervals are contained in the union of $(k + 2)$ intervals $[[(2k + 1)(2\log n + k + 2) + k]] \cup [n - (2k + 1)(2\log n + k + 2) - k + 1, n] \cup (\cup_{i=1}^{k}[\delta_i - \lfloor[(2k + 1)(2\log n + k + 2) + k]/2\rfloor, \delta_i + \lfloor[(2k + 1)(2\log n + k + 2) + k]/2\rfloor])$.

Furthermore, note that there exists at least one deletion location δ_i between two different marked intervals. Otherwise the unmarked interval in between is marked. Therefore, there are at most $k+2$ unmarked intervals. Hence, we conclude that the output consists of at most $k+2$ unmarked intervals, each with length at most $(2k+1)(2\log n + k + 2) + k$, that contain all deletion locations. The proof is done. □

Recall the matrix representation U of the q-ary codeword $\mathbf{u}$. In light of Lemma 8.3.1, we protect the first row U_1^r from k deletions and use it to determine the ranges where the deletions occur. Since the deletion ranges have short length, most of the symbols in sequence $\mathbf{u} \in [[q]]^n$ can be recovered. To this end, the first row U_1^r is generated such that $U_1^r \in \mathcal{L}(n,k)$. Define the set

$$\mathcal{L}^q(n,k) = \{\mathbf{u} : \mathbf{u} \in [[q]]^n, U_1^r \in \mathcal{L}(n,k)\}.$$

The following Lemma shows how to generate sequences in $\mathcal{L}^q(n,k)$.

Lemma 4.6.4. *For any $\mathbf{u} \in [[q]]^n$, there exists a seed s of $O(\log n)$ bits and a function $T_1^q(\mathbf{u}, s) : [[q]]^n \times \{0,1\}^{O(\log n)} \to [[q]]^n$, computable in $poly(n,k)$ time, such that $T_1^q(\mathbf{u}, s) \in \mathcal{L}^q(n,k)$.*

Proof. According to Lemma 7.2.4, there exists a seed s of $O(\log n)$ bits, computable in $poly(n,k)$ time, such that $g_1(s) + U_1^r \in \mathcal{L}(n,k)$. Define $T_1^q(\mathbf{u}, s)$ by

$$T_1^q(\mathbf{u}, s) = \begin{bmatrix} g_1(s) + U_1^r \\ U_2^r \\ \dots \\ U_{\lceil \log q \rceil}^r \end{bmatrix}.$$

Then we have that $T_1^q(\mathbf{u}, s) \in \mathcal{L}^q(n,k)$. □

Let $\mathbf{u} \in \mathcal{L}^q(n,k)$ be a sequence and $\boldsymbol{z} = \mathbf{u}(\delta_1, \dots, \delta_k)$ be the length $n-k$ subsequence of $\mathbf{u}$ after deleting the δ_i-th symbol, $i \in [k]$. We can protect U_1^r against k deletions and recover it by using the code in Theorem 4.1.2. Then given U_1^r and its $n-k$ subsequence Z_1^r, it is possible from Lemma 8.3.1 to find $k+2$ intervals $[a_i, b_i], i \in [[k+2]]$ each having length $T \triangleq (2k+1)(2\log n + k + 2) + k$, that contain all deletion locations. Split $\mathbf{u}$ into blocks $\mathbf{u}_i = (u_{(i-1)T+1}, \dots, u_{iT}), i \in [[\lceil m/T \rceil]]$, of length T. Then the interval $[a_i, b_i], i \in [[k+2]]$, covers at most two blocks in $\mathbf{u}$. Note that the symbol $\mathbf{u}_j$ for $j \in [[n]] \backslash (\cup_{i=1}^{k+2} [a_i, b_i])$ can be determined by

$$\mathbf{u}_j = \mathbf{z}_{j-N_j}, \tag{4.14}$$

where N_j is obtained from Lemma 8.3.1. Hence there are at most $2k+4$ block errors in $\mathbf{u}$ after recovering U_1^r.

Next, we show how to correct the block errors. The idea is to represent each block using its symbol frequency and the symbol location. Specifically, for a q-ary sequence $\mathbf{u} \in [[q]]^n$, define its histogram vector $H(\mathbf{u}) : [[q]]^n \to [[n+1]]^q$ by

$$H(\mathbf{u})_i = |\{j : \mathbf{u}_j = i, j \in [[n]]\}|, i \in [[q]], \tag{4.15}$$

where the i-th entry of $H(\mathbf{u})$ is the number of occurrence of $i \in [[q]]$ in $\mathbf{u}$. Its location vector $V(\mathbf{u}) : [[q]]^n \to [n]^n$ is defined by

$$V(\mathbf{u})_i = \text{the index of the } i\text{-th largest symbol in } \mathbf{u}, \tag{4.16}$$

where a symbol u_i is larger than u_j, if u_i is lexicographically larger than u_j or if $u_i = u_j$ and $i > j$. Note that by definition, we have that $u_{V(\mathbf{u})_1} > u_{V(\mathbf{u})_2} > \ldots > u_{V(\mathbf{u})_n}$. The following lemma shows that a sequence $\mathbf{u} \in [[q]]^n$ is uniquely determined by its histogram and the location vectors.

Lemma 4.6.5. *Let* $\mathbf{u}, \mathbf{y} \in [[q]]^n$ *be two sequences. If* $H(\mathbf{u}) = H(\mathbf{y})$ *and* $V(\mathbf{u}) = V(\mathbf{y})$*, then* $\mathbf{u} = \mathbf{y}$.

Proof. The lemma follows from the definition Eq. (4.15) and Eq. (4.16). Note that the vector $\mathbf{u}$ can be obtained by sequentially putting the i-th largest symbol in $H(\mathbf{u})$ in location $V(\mathbf{u})_i$. Hence the same histogram and location vectors result in the same sequence. □

Next we show how to protect the histogram and location vectors of $\mathbf{u}$ from block errors. Let the block histogram vector $BH(\mathbf{u}) : [[q]]^n \to \left([[T+1]]^q\right)^{\lceil n/T \rceil}$ of the sequence $\mathbf{u} \in [[q]]^n$ be given by

$$BH(\mathbf{u})_i = H(\mathbf{u}_i), i \in [\lceil n/T \rceil], \tag{4.17}$$

where the i-th entry of $BH(\mathbf{u})$ is the histogram vector of the i-th block in $\mathbf{u}$, $i \in [[\lceil n/T \rceil]]$. Similarly, define the block location vector $BV(\mathbf{u}) : [[q]]^n \to \left([T]^T\right)^{\lceil n/T \rceil}$ by

$$BV(\mathbf{u})_i = V(\mathbf{u}_i), i \in [\lceil n/T \rceil], \tag{4.18}$$

where the i-th entry $BV(\mathbf{u})_i$ is the location vector of the i-th block in $\mathbf{u}$. According to Lemma 4.6.5, the sequence $\mathbf{u}$ can be uniquely determined by $BH(\mathbf{u})$ and $BV(\mathbf{u})$.

Define the function $f^{BH}(\mathbf{u}) = (RS_{2k+4}(RS_{2k}(BH(\mathbf{u})_1), \ldots, RS_{2k}(BH(\mathbf{u})_{\lceil \frac{n}{T} \rceil}))$, which is the redundancy of a systematic Reed-Solomon code correcting $2k+4$ erasure errors that has length $\lceil \frac{n}{T} \rceil$ sequence with entries $RS_{2k}(BH(\mathbf{u})_i)$, $i \in [[\lceil n/T \rceil]]$. Each entry $RS_{2k}(BH(\mathbf{u})_i)$ represents the redundancy of a systematic Reed Solomon code correcting k substitution errors in the length q sequence $H(\mathbf{u}_i)$ for $i \in [[\lceil m/T \rceil]]$. The next lemma shows that $f^{BH}(\mathbf{u})$ can be used to protect $BH(\mathbf{u})$. In the following, the function $f_k = f_k^{sys}$ is from Lemma 4.4.4.

Lemma 4.6.6. *Let $\mathbf{u}, \mathbf{y} \in \mathcal{L}^q(m,k)$ be two sequences such that $\mathbf{y} \in \mathcal{B}_k^q(\mathbf{u})$. If $f_k(U_1^r) = f_k(Y_1^r)$ and $f^{BH}(\mathbf{u}) = f^{BH}(\mathbf{y})$, then $BH(\mathbf{u}) = BH(\mathbf{y})$.*

Proof. Since $f_k(U_1^r) = f_k(Y_1^r)$, and $U_1^r \in \mathcal{B}_k(Y_1^r)$, we have that $U_1^r = Y_1^r$. Let $\boldsymbol{z} = \mathbf{u}(\delta_1, \ldots, \delta_k)$. Then according to Lemma 8.3.1, we can obtain from U_1^r and Z_1^r at most $k+2$ intervals $[a_i, b_i], i \in [[k+2]]$, of length at most T, such that $\{\delta_1, \ldots, \delta_k\} \subset \cup_{i=1}^{k+2}[a_i, b_i]$. The intervals $\{[a_i, b_i]\}_{i=1}^{k+2}$ cover at most $2k+4$ blocks in $\mathbf{u}$. Moreover, for any $j \in [[n]] \setminus \cup_{i=1}^{k+2} [a_i, b_i]$, the number $N_j = [[j-1]] \cap \{\delta_1, \ldots, \delta_k\}$ can be determined. Then from Eq. (4.14), there are at most $2k+4$ block errors left, the indices of which are known. Hence given $f^{BH}(\mathbf{u})$, we can recover the sequence $(RS_{2k}(BH(\mathbf{u})_1), \ldots, RS_{2k}(BH(\mathbf{u})_{\lceil m/T \rceil}))$.

Next, we show that the histogram vector $BH(\mathbf{u})_i = H(\mathbf{u}_i)$ can be recovered given $RS_{2k}(BH(\mathbf{u})_i)$ and $\boldsymbol{z}$. Note that $(z_{(i-1)T+1}, \ldots, z_{iT-k})$ is a $T-k$ subsequence of the i-th block $\mathbf{u}_i$, hence the histogram vector $H((z_{(i-1)T+1}, \ldots, z_{iT-k}))$ differs in at most k locations from $H(\mathbf{u}_i)$. Hence given $RS_{2k}(BH(\mathbf{u})_i)$ and $(z_{(i-1)T+1}, \ldots, z_{iT-k})$, we can recover $H(\mathbf{u}_i) = BH(\mathbf{u})_i$ for $i \in [[\lceil n/T \rceil]]$. This implies that if $\mathbf{u}$ can be uniquely recovered given $\mathbf{z}$, U_1^r, and $f^{BH}(\mathbf{u})$. Hence the proof is done. □

We now show how to protect the block location vector $BV(\mathbf{u})$. Let

$$f^{BV}(\mathbf{u}) = RS_{2k+4}(BV(\mathbf{u})) \tag{4.19}$$

be the redundancy of a systematic Reed-Solomon code correcting $2k+4$ erasure errors in the length $\lceil n/T \rceil$ sequence $BV(\mathbf{u})$ with entries $BV(\mathbf{u})_i = V(\mathbf{u}_i)$ for $i \in [[\lceil n/T \rceil]]$ (see Eq. (4.18)). The next lemma shows that $f^{BV}(\mathbf{u})$ can be used to recover $BV(\mathbf{u})$.

Lemma 4.6.7. *For sequences $\mathbf{u}, \mathbf{y} \in \mathcal{L}^q(n,k)$ such that $\mathbf{y} \in \mathcal{B}_k^q(\mathbf{u})$, if $f_k(U_1^r) = f_k(Y_1^r)$ and $f^{BV}(\mathbf{u}) = f^{BV}(\mathbf{y})$, then $BV(\mathbf{u}) = BV(\mathbf{y})$.*

Proof. By similar arguments to that in the proof of Lemma 4.6.6, it can be shown that all but at most $2k+4$ blocks in $\mathbf{u}$ can be recovered, given $f_k(U_1^r)$ and $\boldsymbol{z} = \mathbf{u}(\delta_1, \dots, \delta_k)$. Hence the sequence $BV(\mathbf{u})$ can be recovered with at most $2k+4$ block errors, the indices of which are known. Then $BV(\mathbf{u})$ can be corrected using the Reed-Solomon code redundancy $f^{BV}(\mathbf{u})$. □

Now we are ready to define the labeling function $f_k^{q_2}(\mathbf{u})$ for $\mathbf{u} \in \mathcal{L}^q(n,k)$. Let

$$f_k^{q_2}(\mathbf{u}) = (f_k(U_1^r), f^{BH}(\mathbf{u}), f^{BV}(\mathbf{u})).$$

Then from Lemma 4.4.4, Lemma 8.3.1, Lemma 4.6.5, Lemma 4.6.6, and Lemma 4.6.7, we have the following lemma.

Lemma 4.6.8. *For two sequences $\mathbf{u}, \mathbf{y} \in \mathcal{M}^q(n,k)$, if $\mathbf{y} \in \mathcal{B}_k^q(\mathbf{u})$, then $f_k^{q_2}(\mathbf{u}) \neq f_k^{q_2}(\mathbf{y})$.*

The image of the labeling function $f_k^{q_2}(\mathbf{u})$ consists of R^q bits where

$$\begin{aligned} R^q =& O(k^2 \log k \log n) + (2k+4)T\lceil \log T \rceil + (2k+4)(2k \max\{\log T, \log O(q)\}) \\ =& O((2k+4)\log\log n \cdot \log n), \end{aligned}$$

where the term $(2k+4)T\lceil \log T \rceil$ comes from $f^{BV}(\mathbf{u})$. The term $(2k+4)(2k \max\{\log T, \log O(q)\})$ comes from $f^{BH}(\mathbf{u})$, where $2k \max\{\log T, \log O(q)\}$ is the size of $R_{2k}(BH(\mathbf{u})_i)$, $i \in [[\lceil n/T \rceil]]$. Since $f_k^{q_2}(\mathbf{u})$ does not have the redundancy property, we apply the syndrome compression technique twice, as we did in Case (1).

Define that set

$$\mathcal{D}(\mathbf{u}) = \left\{ j : j > 0, j | (f_k^{q_2}(\mathbf{u}) - f_k^{q_2}(\mathbf{y})) \text{ for some } \mathbf{y} \in \mathcal{L}^q(n,k) \cap \mathcal{B}_k^q(\mathbf{u}) \right\}$$

for $\mathbf{u} \in \mathcal{L}^q(n,k)$. Then from Lemma 4.2.1, we have that

$$|\mathcal{D}(\mathbf{u})| \leq 2^{\log |\mathcal{B}_k^q(\mathbf{u})| + \frac{1.6}{\log e} \cdot \frac{R^q}{\ln \frac{R^q}{\log e}}} \leq 2^{2k \log n + k \log q + \frac{1.6}{\log e} \cdot \frac{R^q}{\ln \frac{R^q}{\log e}}} = 2^{O(poly(k)\log n)}.$$

Hence there exists an integer $\alpha_{\mathbf{u}} \leq 2^{O(poly(k)\log n)}$ such that $f_k^{q_2}(\mathbf{u}) \not\equiv f_k^{q_2}(\mathbf{y}) \bmod \alpha_{\mathbf{u}}$ for $\mathbf{u}, \mathbf{y} \in \mathcal{L}^q(n,k)$ and $\mathbf{y} \in \mathcal{B}_k^q(\mathbf{u})$. Define the labeling function

$$f_k^{q_3}(\mathbf{u}) = (f_k^{q_2}(\mathbf{u}) \bmod \alpha_{\mathbf{u}}, \alpha_{\mathbf{u}})$$

for $\mathbf{u} \in \mathcal{L}^q(n,k)$. Then we have that $f_k^{q_3}(\mathbf{u}) \neq f_k^{q_3}(\mathbf{y})$ for $\mathbf{u}, \mathbf{y} \in \mathcal{M}^q(m,k)$ and $\mathbf{y} \in \mathcal{B}_{k,q}(\mathbf{u})$. Since $f_k^{q_3}(\mathbf{u})$ satisfies the redundancy property, we can use the syndrome compression technique again and find an integer $\alpha \leq 2^{|\mathcal{B}_k^q(\mathbf{u})|+o(\log n)} = 2^{2\log n+\log q+o(\log n)}$ such that $f_k^{q_3}(\mathbf{u}) \not\equiv f_k^{q_3}(\mathbf{y}) \bmod \alpha$ for $\mathbf{u}, \mathbf{y} \in \mathcal{L}^q(n,k)$ and $\mathbf{y} \in \mathcal{B}_k^q(\mathbf{u})$.

Define the code $C(n,k,q)$ as follows:

$$C(n,k,q) = \Big\{\mathbf{x} = \Big(T_1^q(\mathbf{u},s), f_k^{q_3}(T_1^q(\mathbf{u},s)) \bmod \alpha, \alpha, s,$$
$$Hash_k\big(f_k^{q_3}(T_1^q(\mathbf{u},s)) \bmod \alpha, \alpha, s\big)\Big) \in \{0,1\}^n : \mathbf{u} \in \{0,1\}^k\Big\}.$$

Since $f_k^{q_3}(T_1^q(\mathbf{u},s))$ satisfies the redundancy property, it follows from syndrome compression that the redundancy $(f_k^{q_3}(T_2^q(\mathbf{u},s)) \bmod \alpha, \alpha)$ can be described by $2\log \mathcal{B}_k^q(\mathbf{u}) + o(\log n) = 4k\log n + 2k\log q + o(\log n)$ bits. The seed s has length $O(\log k)$. Therefore, the total redundancy is $4k\log n + O(\log n) + 2k\log q + o(\log n)$ bits. The correctness of the code can be proved by the same argument as in the proof of Theorem 4.6.1.

Theorem 4.6.2. *The code $C(n,k,q)$ is a k deletion code.*

4.7 q-ary Codes Correcting k Deletions for Large q

In this section, we consider the problem of coding for deletions over large non-binary alphabets. We will show that in this regime we can construct efficiently encodable/decodable codes capable of correcting k deletions that require roughly $30k\log n$ bits of redundancy. The approach taken in this section is fundamentally different than the syndrome compression technique that has been used up to this point. Note that, compared to the syndrome compression technique, the redundancy of our code is high. However, the advantage of the approach discussed here is that our methods are more applicable to a wider range of k. In particular, the technique described here, which is similar in spirit to the approach taken in [98] to correct errors in permutations, has decoding/encoding complexity which scales polynomially for any k and, in addition, leads to efficiently encodable/decodable codes for the regime where k is a small constant fraction of n.

We will construct codes by making use of the L-spectrum, which represents the set of all length L subsequences of consecutive symbols that appear in a vector. For a vector $\mathbf{u} \in [[q]]^n$ (where $q > n$), we denote the L-spectrum for $\mathbf{u}$, denoted $S_L(\mathbf{u})$ as follows:

$$S_L(\mathbf{u}) = \Big\{(u_i, u_{i+1}, \ldots, u_{i+L-1}) \in [[q]]^L : i \in [n-L+1]\Big\}.$$

Furthermore, we say that a sequence $\mathbf{u} \in [[q]]^m$ is L-substring unique if

$$|S_L(\mathbf{u})| = n - L + 1.$$

The approach taken to correcting deletions is motivated by the following two lemmas, the first of which also appears in [33].

Lemma 4.7.1. *(c.f., [33]) Suppose* $\mathbf{u} \in [[q]]^n$ *is* $(L-1)$*-substring unique. Then,* $\mathbf{u}$ *can be uniquely recovered from* $S_L(\mathbf{u})$*.*

For shorthand, for two sets A, B let $A \triangle B = (A \setminus B) \cup (B \setminus A)$ denote their symmetric difference.

Lemma 4.7.2. *Suppose* $\mathbf{u} \in [[q]]^n$ *is* $(L-1)$*-substring unique and* $\mathbf{z} \in [[q]]^{n-k}$ *is the result of* k *deletions occurring to* $\mathbf{u}$*. Then,*

$$|S_L(\mathbf{u}) \triangle S_L(\mathbf{z})| \leq (2L-1)k.$$

Proof. Notice that for each deletion in $\mathbf{u}$, $|S_L(\mathbf{u}) \setminus S_L(\mathbf{z})| \leq L$ and $|S_L(\mathbf{z}) \setminus S_L(\mathbf{u})| \leq L-1$. Since there are k deletions the result follows. □

In light of the previous two lemmas, our approach will consist of two basic steps:

1. **Transform step**: In this step, we convert our information vectors into vectors which are L-substring unique.
2. **Coding step**: We add additional redundancy symbols to our codewords to ensure that we can recover their L-spectrum provided deletion errors are allowed to occur.

We begin by describing some results that are related to the transform step. Define $\mathcal{U}_L^q(n)$ so that

$$\mathcal{U}_L^q(n) = \left\{ \mathbf{u} \in [[q]]^n : \mathbf{u} \text{ is } L\text{-substring unique} \right\}.$$

The following result provides an algorithm to generate binary sequences in $\mathcal{U}_L^q(n)$ for $L = 2\log n + 2$. This algorithm will be used in Lemma 4.7.4 to generate non-binary sequences that are L-substring unique.

Lemma 4.7.3. *[28], There exists an invertible function $h_L : \{0,1\}^{n-1} \to \{0,1\}^n$, computable in $poly(n)$ time, that takes any binary sequence $\mathbf{u} \in \{0,1\}^{n-1}$ as input and outputs a sequence $h_L(\mathbf{u}) \in \mathcal{U}^2_L(n)$ for $L = 2\log n + 2$.*

Lemma 4.7.3 can be used to generate sequences in $\mathcal{U}^q_3(n)$.

Lemma 4.7.4. *There exists an invertible function $h_L : [[q]]^n \to [[q]]^{n+1}$, computable in $poly(n)$ time, such that $h_L(\mathbf{u}) \in \mathcal{U}^q_L(n)$ for any $\mathbf{u} \in [[q]]^n$ where $L = 3$.*

Proof. Let $\mathbf{w} \in \{0,1\}^{n\log q}$ be the binary representation of $\mathbf{u}$, i.e., $(w_{(i-1)\log q+1}, \dots, w_{i\log q}) \in \{0,1\}^{\log q}$ is the binary representation for $u_i \in [[q]]$ for $i \in [n]$. Then from Lemma 4.7.3 we have that $h_L((\mathbf{w}, \mathbf{0}^{\log q-1})) \in \mathcal{U}^2_L((n+1)\log q))$ for $L = 2\log(n+1) + 2\log\log q + 2$, where $(\mathbf{w}, \mathbf{0}^{\log q-1})$ is obtained by adding $\log q - 1$ 0's to the end of $\mathbf{w}$. Split $h_L((\mathbf{w}, \mathbf{0}^{\log q-1}))$ into $n+1$ blocks $\mathbf{h}_i = (h_L(\mathbf{w})_{(i-1)\log q+1}, h_L(\mathbf{w})_{(i-1)\log q+2}, \dots, h_L(\mathbf{w})_{i\log q})$, $i \in [n+1]$, of length $\log q$. Let $h_L(\mathbf{u})_i$ be the q-ary representation of $\mathbf{h}_i$, $i \in [n+1]$.

We now prove by contradiction that $h_L(\mathbf{u}) \in \mathcal{U}^q_3(n+1)$ for n sufficiently large enough. Suppose on the contrary, we have that $(h_L(\mathbf{u})_i, h_L(\mathbf{u})_{i+1}, h_L(\mathbf{u})_{i+2}) = (h_L(\mathbf{u})_j, h_L(\mathbf{u})_{j+1}, h_L(\mathbf{u})_{j+2})$ for some $i \neq j$. Then we have that

$$\begin{aligned}
&(h_L((\mathbf{w}, \mathbf{0}^{\log q-1}))_{(i-1)\log q+1}, h_L((\mathbf{w}, \mathbf{0}^{\log q-1}))_{(i-1)\log q+2}, \\
&\quad \dots, h_L((\mathbf{w}, \mathbf{0}^{\log q-1}))_{(i+2)\log q}) \qquad (4.20)\\
=&(h_L((\mathbf{w}, \mathbf{0}^{\log q-1}))_{(j-1)\log q+1}, h_L((\mathbf{w}, \mathbf{0}^{\log q-1}))_{(j-1)\log q+2}, \qquad (4.21)\\
&\quad \dots, h_L((\mathbf{w}, \mathbf{0}^{\log q-1}))_{(j+2)\log q}). \qquad (4.22)
\end{aligned}$$

Since $q > n$, we have that

$$3\log q > 2\log q + 2\log\log q + 2 \geq 2\log(n+1) + 2\log\log q + 2$$

for n sufficiently large. Hence from Eq. (4.20), we have that

$$\begin{aligned}
&(h_L((\mathbf{w}, \mathbf{0}^{\log q-1}))_{(i-1)\log q+1}, h_L((\mathbf{w}, \mathbf{0}^{\log q-1}))_{(i-1)\log q+2}, \dots, \\
&h_L((\mathbf{w}, \mathbf{0}^{\log q-1}))_{(i-1)\log q+2\log(n+1)+2\log\log q+2}) \\
=&(h_L((\mathbf{w}, \mathbf{0}^{\log q-1}))_{(j-1)\log q+1}, h_L((\mathbf{w}, \mathbf{0}^{\log q-1}))_{(j-1)\log q+2}, \dots, \\
&h_L((\mathbf{w}, \mathbf{0}^{\log q-1}))_{(j-1)\log q+2\log(n+1)+2\log\log q+2}),
\end{aligned}$$

which is a contradiction to the fact that $h_L((\mathbf{w}, \mathbf{0}^{\log q-1})) \in \mathcal{U}^2_L((n+1)\log q)$ for $L = 2\log(n+1) + 2\log\log q + 2$. Hence, we have that $(h_L(\mathbf{u})_i, h_L(\mathbf{u})_{i+1}, h_L(\mathbf{u})_{i+2}) \neq$

$(h_L(\mathbf{u})_j, h_L(\mathbf{u})_{j+1}, h_L(\mathbf{u})_{j+2})$ for $i \neq j$. Moreover, since $h_L((\mathbf{w}, \mathbf{0}^{\log q-1}))$ is invertible, we can recover $\mathbf{w}$ and thus $\mathbf{u}$ from $h_L(\mathbf{u})$. Therefore, the proof is done. □

Now, we turn to describing the coding step (step 2)) of our construction. We will interpret the 4-spectrum of our codewords using indicator vectors. In particular, define the 4-profile indicator vector $\mathbb{1}_4^q(\boldsymbol{u}) \in \{0,1\}^{q^4}$, which is indexed by the non-zero elements in $[[q]]^4$, by

$$\mathbb{1}_4^q(\boldsymbol{u})_{\mathbf{z}} = \begin{cases} 1 & \text{if } \mathbf{z} \in S_4(\mathbf{u}), \\ 0 & \text{else} \end{cases}$$

for $\mathbf{z} \in [q^4]$. Note that the indices of the 1 entries in $\mathbb{1}_4^q(\mathbf{u})$ correspond to the 4-spectrum of $\mathbf{u}$.

An immediate consequence of Lemma 4.7.2 is the following.

Corollary 4.7.1. *Suppose* $\mathbf{u} \in [[q]]^n$ *is* 4*-substring unique and* $\mathbf{z} \in [[q]]^{n-k}$ *is the result of* k *deletions occurring to* $\mathbf{u}$*. Then,*

$$d_H(\mathbb{1}_4^q(\mathbf{u}), \mathbb{1}_4^q(\mathbf{z})) \leq 7k,$$

where d_H *denotes the Hamming distance.*

For a vector $\mathbf{v} \in \{0,1\}^{q^4}$, let $\mathrm{BCH}_{7k}(\mathbf{v})$ denote the $28k \log q$ redundant bits from a systematic BCH code of dimension q^4 that is capable of correcting $7k$ substitution errors. The idea now is to encode these $28k \log q$ bits of information (which will be used to protect the indicator vectors for our codewords), into the final $29k + 1$ symbols of our codewords, while reserving a portion of each symbol to store location information.

Let $\mathbf{r}_I = (r_1, r_2, \ldots, r_{29k}) \in [[\frac{q}{30k}]]^{29k}$ be the output of $\mathrm{BCH}_{7k}(\mathbb{1}_4^q(\mathbf{u}))$ represented as $\frac{q}{30k}$-ary symbols. Clearly, we can represent $29k \log \frac{q}{30k}$ bits of information with $\mathbf{r}_I$. Note that this encoding is possible since

$$29k \log \frac{q}{30k} \geq 28k \log q$$

which holds when $\log q \geq 29 \log(30k)$.

Let $RS_{\lfloor \frac{k+1}{2} \rfloor}$ be a Reed-Solomon code over $[[\frac{q}{30k}]]$ that can correct either $\lfloor \frac{k+1}{2} \rfloor$ substitution errors or k erasures and the code has minimum distance at least $k+1$.

We assume $RS_{\lfloor \frac{k+1}{2} \rfloor}$ has dimension $29k$ and that for a vector $\mathbf{v} \in [[\frac{q}{30k}]]$ $RS_{\lfloor \frac{k+1}{2} \rfloor}(\mathbf{v})$ outputs k redundant symbols. Let

$$\mathbf{r} = \left(\mathbf{r}_I, RS_{\lfloor \frac{k+1}{2} \rfloor}(\mathbf{r}_I)\right) \in [[\frac{q}{30k}]]^{30k}.$$

We are now ready to present the k deletion code in terms of the encoding process. Suppose $\mathbf{u} \in [[q]]^n$ is an information vector of dimension n. The output of the encoding process will be a vector $\mathbf{x} \in [[q]]^N$, where $N = n + 1 + 30k$.

1. Suppose $\mathbf{u}_T = h_3(\mathbf{u}) \in [[q]]^{n+1}$ where h_3 is defined in Lemma 4.7.4.
2. Let $\mathbf{r}_I = (r_1, r_2, \ldots, r_{29t}) \in [[\frac{q}{30k}]]^{29k}$ denote the $\frac{q}{30k}$-ary representation of $\text{BCH}_{7k}(\mathbb{1}_4^q(\mathbf{u}_T)) \in \{0,1\}^{28k \log q}$.
3. Let $\mathbf{r} = \left(\mathbf{r}_I, RS_{\lfloor \frac{k+1}{2} \rfloor}(\mathbf{r}_I)\right) = (R_1, R_2, \ldots, R_{30k}) \in [[\frac{q}{30k}]]^{30k}$.
4. Define $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ so that

$$x_i = \begin{cases} (\mathbf{u}_T)_i & \text{if } i \in [n+1], \\ \left(i - (n+1) + 1, (\mathbf{r})_{i-(n+1)+1}\right) & \text{else.} \end{cases}$$

Theorem 4.7.1. *Suppose $\mathbf{x} \in [[q]]^N$ is transmitted and $\mathbf{z} \in [[q]]^{N-k}$ is received where $\mathbf{z}$ is the result of k deletions occurring to $\mathbf{x}$. Then given $\mathbf{z}$ it is possible to uniquely determine $\mathbf{x}$.*

Proof. We prove the result by describing the decoding procedure. Similar to the proof of Theorem 4.6.1, we first recover the final $29k$ symbols in $\mathbf{x}$ which can be obtained after deleting at most k symbols from the sequence

$$\Big((1, R_1), (2, R_2), \ldots, (30k, R_{30k})\Big).$$

Since every symbol of this sequence has its position encoded into it, it follows that we can determine the locations of the deletions from any length $29k$-subsequence of $\Big((1, R_1), (2, R_2), \ldots, (30k, R_{30k})\Big)$. Then, given this position information along with the fact that $\mathbf{r}$ belongs to a code which can correct k erasures, we can recover $\mathbf{r}$. From $\mathbf{r}$ we can recover $\text{BCH}_{7k}(\mathbb{1}_4^q(\mathbf{u}_T))$. From (4.7.1) we can determine $\mathbb{1}_4^q(\mathbf{u}_T)$. Finally, from Lemma 4.7.3, we recover $\mathbf{u}_T$ and from $\mathbf{u}_T$ we can recover $\mathbf{u}$ according to Lemma 4.7.1. □

4.8 Conclusion

In this chapter, we proposed a syndrome compression technique for achieving redundancy twice the Gilbert-Varshamov lower bound. We applied the technique to several variations of deletion channels and proposed constructions of non-systematic and systematic binary deletion correcting codes, codes correcting bursts of deletions, and non-binary deletion correcting codes. Most of the codes improve the best existing redundancy result. It will be of great interest to investigate whether it is possible to reduce the encoding/decoding complexity of the syndrome compression technique, which is currently $n^{O(1)}$.

4.9 Appendix: Generation of Random Seed z with Size $O(\log m)$

In the following, we prove that we can generate strings in the k-mixed string set $\mathcal{M}(m,k)$ using $O(\log m)$ bits. The proof follows similar steps to those in [22]. We first recall the definition of the k-mixed string set

$$\begin{aligned}&\mathcal{M}(m,k)=\{\mathbf{x}\in\{0,1\}^m:\text{ For integers }\ell=\lceil\log k+\log\log(k+1)+5\rceil\text{ and}\\&d=O(k(\log k)^2\log m)\text{ and for any string }\boldsymbol{p}\in\{0,1\}^\ell\text{, every substring of}\\&\text{consecutive }d\text{ bits in }\mathbf{x}\text{ contains }\boldsymbol{p}\text{ as a substring.}\}\end{aligned}$$

We now proceed to proving the following lemma, which appears in Sec. 4.3 as Lemma 4.3.2.

Lemma 4.9.1. *There exists a $poly(m)$ time algorithm for a seed $\boldsymbol{s}$ with length $O(\log m)$ such that for any $\mathbf{u}\in\mathbb{F}_2^m$, $\mathbf{u}+g(\boldsymbol{s})\in\mathcal{M}(m,k)$.*

Split the string $\mathbf{u}$ into $n=m/d$ blocks $\mathbf{u}_i$, $i\in[n]$ of length d. Split each block $\mathbf{u}_i$ into $n_0=O(\log m)$ subblocks $\mathbf{u}_{i,j}$, $j\in[n_0]$ of length $d_0=d/n_0=O(k(\log k)^2)$. Using the random seed $\mathbf{s}$, we generate the same mask for each block $\mathbf{u}_i$. Hence in the following, we focus on generating a random mask for block $\mathbf{u}_1$. The idea is to generate the random seed $\boldsymbol{s}=(\boldsymbol{s}_1,\dots,\boldsymbol{s}_{n_0})$ and the corresponding mask $\boldsymbol{e}=(\boldsymbol{e}_1,\dots,\boldsymbol{e}_{n_0})$ subblock by subblock for each $\mathbf{u}_{1,j}$, $j\in[n_0]$. The goal is that $\boldsymbol{u}_{1,j}+\mathbf{e}_j$ contains $\mathbf{p}$ with at least constant probability c for $j\in[n_0]$. Then the probability that all subblocks $\mathbf{u}_{1,j}$ do not contain $\mathbf{p}$ is upper bounded by $(1-c)^{n_0}=1/poly(m)$. We begin with the following claim.

Claim 4.9.1. *For any string $\boldsymbol{p}\in\{0,1\}^\ell$, a uniformly distributed random string U_{d_0} of length d_0 contains $\boldsymbol{p}$ with probability at least $2/3$.*

Proof. Note that a uniformly random string of length ℓ equals $\boldsymbol{p}$ with probability $1/2^{\ell}$, by properly choosing d_0. Split U_{d_0} into blocks of length ℓ. Then the probability that all these blocks do not equal $\boldsymbol{p}$ is given by $(1 - 1/2^{\ell})^{d_0/\ell} = (1 - 1/2^{\ell})^{O(2^{\ell})} \leq 1/e^{O(1)}$. By properly choosing d_0 we can make this probability less than $1/3$. Then, the probability that U_{d_0} contains $\boldsymbol{p}$ is at least $2/3$. □

By virtue of Claim 4.9.1, if we generate $\mathbf{s}_j = U_{d_0}$ independently and let the mask $\mathbf{e}_j = \mathbf{s}_j$ for $j \in [n_0]$. Then $\mathbf{u}_{1,j} + \mathbf{e}_j$ contains $\mathbf{p}$ with probability at least $c = 2/3$. The probability that $\mathbf{u}_1$ does not contain $\mathbf{p}$ is upper bounded by $(1/3)^{n_0} = 1/poly(m)$. However, this requires the seed length to be $n_0 d_0 = k \log^2 k \log m$, which is larger than $O(\log m)$. To reduce the size of the seed, we generate the seed $\mathbf{s}_j$ and the mask $\mathbf{e}_j$ by random walk on expander graphs. For reference we restate the definitions and the results below. Lemma 4.9.2 and Lemma 4.9.3 are also cited in [22].

Definition 4.9.1. *If G is a q-regular graph with n vertices and $\lambda(G) \leq \lambda$ for integers n and q and real number $\lambda < 1$, we say that G is an (n, q, λ)-expander graph. Here $\lambda(G)$ is the second largest eigenvalue of the normalized adjacency matrix of G.*

For some constant integer q and some number $\lambda < 1$, a family of graphs $\{G_n\}_{n=1}^{\infty}$ is a (q, λ)-expander graph family if for every $n \in \mathbb{N}$, G_n is an (n, q, λ)-expander graph.

Lemma 4.9.2. *Let $A_0, \ldots, A_s$ be vertex sets with densities $\alpha_0 = |A_0|/n, \ldots, \alpha_s = |A_s|/n$ in an (n, q, λ)-expander graph G. Let $X_0, \ldots, X_s$ be a random walk on G. Then we have that*

$$Pr[\forall i \in \{0, \ldots, s\}, X_i \in A_i] \leq \prod_{i=0}^{s-1} (\sqrt{\alpha_i \alpha_{i+1}} + \lambda).$$

Lemma 4.9.3. *For every constant $\lambda < 1$, and some constant integer q depending on λ, there exists a strongly explicit (q, λ)-expander graph family (can be provided in $poly(\log n)$ time).*

According to Lemma 4.9.3, we can generate a $(2^{d_0}, q, \lambda = 1/6)$-expander graph G, where q is a constant integer depending on λ. Instead of generating the seed $\boldsymbol{s}_j$ independently for each $j \in [n_0]$, we do a n_0-step random walk on the graph G. We show that the n_0 step random walk on G can be generated using $O(\log m)$ bits. Let $\mathbf{s}_0 = U_{d_0}$ be a uniformly and randomly chosen vertex on the graph G. The vertex $\boldsymbol{s}_0$ is the starting point of the random walk. Since each vertex is connected to a constant q number of vertices, the j-th step can be generated using a $\log q$ bit random seed $\mathbf{s}_j$, $j \in [n_0]$, indicating which vertex to go in the j-th step. Hence the total number of bits needed is $d_0 + \log qn_0 = O(\log m)$. Let $E_0, \ldots, E_{n_0}$ be the trace of the random walk, where $E_j \in \{0,1\}^{d_0}$ is a vertex of the graph G. We use $\mathbf{e}_j = E_j$ as the mask for subblock $\mathbf{u}_{1,j}$.

We are now able to prove our main result in this section.

Proof of Lemma 4.3.2: Let $E_0, E_1, \ldots, E_{n_0}$ be a random walk on the $(2^{d_0}, q, \lambda = 1/6)$-expander graph G. Recall, that n_0 represents the number of subblocks in each block of $\mathbf{x}$. For a string $\boldsymbol{p} \in \{0,1\}^{\ell}$ and a fixed $j \in [n_0]$, define the set $A_j^{\boldsymbol{p}}$, $j \in [n_0]$ to be the set of vertices in G such that for any vertex $\mathbf{s}'$ in this set, the sequence $\mathbf{u}_{1,j} + \mathbf{s}'$ does not contain $\boldsymbol{p}$. Then according to Claim 4.9.1, we have that $\alpha_i = |A_j^{\boldsymbol{p}}|/n \le 1/3$. From Lemma 4.9.2, the probability that $E_j \in A_j^{\mathbf{p}}$, i.e., $E_j + \mathbf{u}_{1,j}$ does not contain $\mathbf{p}$, for all $j \in [n_0]$ is at most

$$(1/3 + 1/6)^{n_0} = (1/2)^{O(\log n)} = 1/poly(m).$$

Let $g'(\boldsymbol{s}) = (E_1, \ldots, E_{n_0})$. Recall that $\boldsymbol{s}$ is the length $O(\log m)$ random seed that generates the random walk $E_0, E_1, \ldots, E_{n_0}$. Then the probability that $\mathbf{u}_1 + g'(\mathbf{s})$ does not contain $\mathbf{p}$ is at most $1/poly(m)$. Similarly, $\mathbf{u}_i + g'(\mathbf{s})$, $i \in [n]$ does not contain $\mathbf{p}$ with probability at most $1/poly(m)$. Hence from the union bound, the probability that there exists a block $\mathbf{u}_i$ such that $\boldsymbol{u}_i + g'(\mathbf{s})$ does not contain $\boldsymbol{p}$ for some $i \in [n]$ and some $\boldsymbol{p} \in \{0,1\}^{\ell}$ is upper bounded by

$$2^{\ell} n/poly(m) = 1/poly(m)$$

by choosing n_0 to be sufficiently large enough.

Let $g(\boldsymbol{s})$ be the $n = m/d$-fold repetition of $g'(\boldsymbol{s})$. Then we conclude that $\mathbf{u} \oplus g(\mathbf{s})$ belongs to $\mathcal{M}(m,k)$ with probability $1 - 1/poly(m)$. Thus, it suffices to search in $2^{O(\log m)} = poly(m)$ time for a seed $\mathbf{s}$ with length $O(\log m)$ such that $\mathbf{u} + g(\mathbf{s}) \in \mathcal{M}(m,k)$. This completes the proof.

We now proceed to the proof of Corollary 4.3.1.

Proof. (of Corollary 4.3.1) The proof is similar to that of Lemma 4.3.2. By Theorem 4.3.1, it suffices to show that $\mathbf{u} \in \mathcal{M}(n,k)$ with high probability. For any sequence $\mathbf{p} \in \{0,1\}^{\ell}$ and integers i and j such that $i + j - 1 \le n$, let $A(\mathbf{p}, i, j)$ be the event that $\mathbf{u}_i, \ldots, \mathbf{u}_{i+j-1}$ does not contain $\mathbf{p}$, where $\ell = \lceil \log k + \log\log(k+1) + 5 \rceil$

is given in the definition of $\mathcal{M}(n, k)$. According to Claim 4.9.1, the probability of $A(\mathbf{p}, i, d_0)$ is at most $1/3$ for sufficiently large $d_0 = O(k \log^2 k)$ and for any $\mathbf{p}$ and $i \in [n - d + 1]$. Hence the probability of $A(\mathbf{p}, i, d_0 n_0)$, where $n_0 = O(\log n)$, is at most $(1/3)^{n_0} = 1/poly(n)$ for any $\mathbf{p}$ and $i \in [n - d_0 n_0 + 1]$.

Then by the union bound, the probability of $\cup_{\mathbf{p} \in \{0,1\}^\ell, i \in [n - d_0 n_0 + 1]}$ is upper bounded by $2^\ell n/poly(n)$, which is $1/poly(n)$ for sufficiently large n_0. Therefore, the probability that $\boldsymbol{u} \notin \mathcal{M}(n, k)$ is at most $1/poly(n)$, which goes to zero as n grows to infinity. Thus the corollary is proved. $\square$

Chapter 5

CODING OVER SETS FOR DNA STORAGE: SUBSTITUTION ERRORS

In this chapter, we study codes over unordered sets of sequences, that correct substitution errors.

5.1 Introduction

In the model of coding over sets, the data to be stored is encoded as a set of M strings of length L over a certain alphabet, for some integers M and L such that $M < 2^L$; typical values for M and L are currently within the order of magnitude of 10^7 and 10^2, respectively [73]. Each individual string is subject to various types of errors, such as deletions (i.e., omissions of symbols, which result in a shorter string), insertions (which result in a longer string), and substitutions (i.e., replacements of one symbol by another).

One of the reasons for error correction is that errors in synthesis might cause the PCR process to amplify a string that was written erroneously, and hence the reconstructed origins might include this erroneous string. In some cases, error correction after synthesis is possible, and yet our model in this chapter is most suitable for substitution errors that were amplified by the PCR process. Deletions and insertions will be discussed in the next chapter. The work in [62] provided a scheme that efficiently corrects a single deletion, which is easier to handle since a single deletion only results a change in the sequence length. Substitution errors, however, are more challenging to combat, and are discussed next.

A substitution error that occurs prior to amplification by PCR can induce either one of two possible error patterns. In one, the newly created string already exists in the set of strings, and hence, the reconstructed origins will constitute a set of $M - 1$ strings. In the other, which is undetectable by counting the size of the set of reconstructed origins, the substitution generates a string which is not equal to any other string in the set. In this case the output set has the same size as the error free one. These error patterns, which are referred to simply as *substitutions*, are the main focus of this chapter.

Following a formal definition of the channel model in Sec. 5.2, previous work is discussed in Sec. 5.3. Upper and lower bounds on the amount of redundant bits that are required to combat substitutions are given in Sec. 5.4. In Sec. 5.5 we provide a construction of a code that can correct a single substitution. The redundancy of this construction is shown to be optimal up to some constant, which is later improved in Appendix 5.8. In Sec. 5.6 the construction for a single substitution is generalized to multiple substitutions, and is shown to be order-wise optimal whenever the number of substitutions is a constant. Finally, open problems for future research are discussed in Sec. 5.7.

Remark 5.1.1. *The channel which is discussed in this chapter can essentially be seen as taking a string of a certain length N as input. Then, during transmission, the string is sliced into substrings of equal length, and each substring is subject to substitution errors in the usual sense. Moreover, the order between the slices is lost during transmission, and they arrive as an unordered set.*

It follows from the sphere-packing bound [79, Sec. 4.2] that without the slicing operation, one must introduce at least $k\log(N)$ redundant bits at the encoder in order to combat k substitutions. The surprising result of this chapter is that the slicing operation does not *incur a substantial increase in the amount of redundant bits that are required to correct these k substitutions. In the case of a single substitution, our codes attain an amount of redundancy that is asymptotically equivalent to the ordinary (i.e., unsliced) channel, whereas for a larger number of substitutions we come close to that, but prove that a comparable amount of redundancy is achievable.*

5.2 Preliminaries

To discuss the problem in its most general form and illustrate the ideas, we restrict our attention to binary strings. Most of the techniques in this chapter can be extended to non-binary cases. Generally, we denote scalars by lower-case letters $x, y, \ldots$, vectors by bold symbols $\mathbf{x}, \mathbf{y}, \ldots$, integers by capital letters $k, L, \ldots$, and $[k] \triangleq \{1, 2, \ldots, k\}$. For integers M and L such that[1] $M \leq 2^L$ we denote by $\binom{\{0,1\}^L}{M}$ the family of all subsets of size M of $\{0,1\}^L$, and by $\binom{\{0,1\}^L}{\leq M}$ the family of subsets of size *at most* M of $\{0,1\}^L$. In our channel model, a *word* is an element $W \in \binom{\{0,1\}^L}{M}$, and a *code* $C \subseteq \binom{\{0,1\}^L}{M}$ is a set of words (for clarity, we refer to words in a given code as *codewords*). To prevent ambiguity with classical coding theoretic terms, the elements in a word $W = \{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$ are referred to as *strings*. We emphasize

[1]We occasionally also assume that $M \leq 2^{cL}$ for some $0 < c < 1$. This is in accordance with typical values of M and L in contemporary DNA storage prototypes (see Sec. 5.1).

that the indexing in W is merely a notational convenience, e.g., by the lexicographic order of the strings, and this information is not available at the decoder.

For $k \le ML$, a *k-substitution error* (k-substitution, in short), is an operation that changes the values of at most k different positions in a word. Notice that the result of a k-substitution is not necessarily an element of $\binom{\{0,1\}^L}{M}$, and might be an element of $\binom{\{0,1\}^L}{T}$ for some $M-k \le T \le M$. This gives rise to the following definition.

Definition 5.2.1. *For a word $W \in \binom{\{0,1\}^L}{M}$, a ball $\mathcal{B}_k(W) \subseteq \bigcup_{j=M-k}^{M} \binom{\{0,1\}^L}{j}$ centered at W is the collection of all subsets of $\{0,1\}^L$ that can be obtained by a k-substitution in W.*

Example 5.2.1. *For $M = 2$, $L = 3$, $k = 1$, and $W = \{001, 011\}$, we have that*

$$\mathcal{B}_k(W) = \{\{001,011\},\{101,011\},\{011\},\{000,011\},\\ \{001,111\},\{001\},\{001,010\}\}.$$

In this chapter, we discuss bounds and constructions of codes in $\binom{\{0,1\}^L}{M}$ that can correct k substitutions (k-substitution codes, for short), for various values of k. The *size* of a code, which is denoted by $|\mathcal{C}|$, is the number of codewords (that is, sets) in it. The *redundancy* of the code, a quantity that measures the amount of redundant information that is to be added to the data to guarantee successful decoding, is defined as $r(\mathcal{C}) \triangleq \log\binom{2^L}{M} - \log(|\mathcal{C}|)$, where the logarithms are in base 2.

A code $\mathcal{C}$ is used in our channel as follows. First, the data to be stored (or transmitted) is mapped by a bijective *encoding function* to a codeword $C \in \mathcal{C}$. This codeword passes through a channel that might introduce up to k substitutions, and as a result a word $W \in \mathcal{B}_k(C)$ is obtained at the decoder. In turn, the decoder applies some *decoding function* to extract the original data. The code $\mathcal{C}$ is called a k-substitution code if the decoding process always recovers the original data successfully. Having settled the channel model, we are now in a position to formally state our contribution.

Theorem 5.2.1. *(Main) For any integers M, L, and k such that $M \le 2^{L/(4k+2)}$, there exists an explicit code construction with redundancy $O(k^2\log(ML))$*[2] *(Sec. 5.6). For $k = 1$, the redundancy of this construction is asymptotically at most six times larger than the optimal one (Sec. 5.5), when L goes to infinity and $M \ge 4$. Furthermore, an improved construction for $k = 1$ achieves redundancy which is asymptot-*

[2]Throughout this chapter, we write $g(n) = O(f(n))$ for any functions $f, g : \mathbb{N} \to \mathbb{R}$ and integer n, if $\limsup_{n\to\infty} \frac{g(n)}{f(n)} < \infty$.

ically at most three times the optimal one (Appendix 5.8), when L goes to infinity and $4 \leq M \leq 2^{L/4}$.

Remark 5.2.1. *We note that under the current technology restriction, our code constructions apply to a limited parameter range. For example, when* $k = 1$ *and* $L = 100$*, our code requires that* $M \leq 10^5$*. The upper bound constraint on M is more strict as k increases. Yet, our constructions provide the following insights.*

First, as the technology advances, it is reasonable to hope that the error rate will decrease and the synthesis length L will increase, in which case we have smaller k and larger L. As k gets smaller, the range of M increases exponentially with L.

Second, while L is limited under current technology, the number of strings M can be more freely chosen. One can get smaller M by storing information in more DNA pools.

Third, the application of this work is not limited to DNA storage. Our techniques apply to other cases such as network packet transmission.

A few auxiliary notions are used throughout this chapter, and are introduced herein. For two strings $\mathbf{s}, \mathbf{t} \in \{0,1\}^L$, the *Hamming distance* $d_H(\mathbf{s}, \mathbf{t})$ is the number of entries in which they differ. To prevent confusion with common terms, a subset of $\{0,1\}^L$ is called *a vector-code*, and the set $\mathcal{B}_D^H(\mathbf{s})$ of all strings within Hamming distance D or less of a given string $\mathbf{s}$ is called the *Hamming ball* of radius D centered at $\mathbf{s}$. A *linear* vector code is called an $[n,k]_q$ code if the strings in it form a subspace of dimension k in $\mathbb{F}_q^n$, where $\mathbb{F}_q$ is the finite field with q elements.

Several well-known vector-codes are used in the sequel, such as Reed-Solomon codes or Hamming codes. For an integer t, the Hamming code is an $[2^t - 1, 2^t - t - 1]_2$ code (i.e., there are t redundant bits in every codeword), and its minimum Hamming distance is 3. Reed-Solomon (RS) codes over $\mathbb{F}_q$ exist for every length n and dimension k, as long as $q \geq n - 1$ [79, Sec. 5], and require $n - k$ redundant symbols in $\mathbb{F}_q$. Whenever q is a power of two, RS codes can be made binary by representing each element of $\mathbb{F}_q$ as a binary string of length $\log_2(q)$. In the sequel we use this form of RS code, which requires $\log(n)(n-k)$ redundant bits.

Finally, our encoding algorithms make use of *combinatorial numbering maps* [53], that are functions that map a number to an element in some structured set. Specifically, $F_{com} : [\binom{N}{M}] \to \{S : S \subset [N], |S| = M\}$ maps a number to a set of distinct elements, and $F_{perm} : [N!] \to S_N$ maps a number to a permutation in the symmetric group S_N. The function F_{com} can be computed using a greedy algorithm with complexity $O(MN \log N)$, and the function F_{perm} can be computed in a straightforward manner with complexity $O(N \log N)$. Using F_{com} and F_{perm} together, we define a map $F : [\binom{N}{M} M!] \to \{S : S \subset [N], |S| = M\} \times S_M$ that maps a number into an unordered set of size M together with a permutation.

5.3 Previous Work

The idea of manipulating atomic particles for engineering applications dates back to the 1950s, with R. Feynman's famous citation, "There's plenty of room at the bottom" [31]. The specific idea of manipulating DNA molecules for data storage as been circulating the scientific community for a few decades, and yet it was not until 2012–2013 where two prototypes have been implemented [24, 36]. These prototypes have ignited the imagination of practitioners and theoreticians alike, and many works followed suit with various implementations and channel models [14, 32, 46, 52, 77, 99].

By and large, all practical implementations to this day follow the aforementioned channel model, in which multiple short strings are stored inside a solution. Normally, deletions and insertions are also taken into account, but substitutions were found to be the most common form of errors [73, Fig. 3.b], and strings that were subject to insertions and deletions are scarcer, and some of them can be corrected using clustering and sequence reconstruction algorithms.

The channel model in this work has been studied by several authors in the past. The work of [46] addressed this channel model under the restriction that individual strings are read in an error-free manner, and some strings might get lost as a result of random sampling of the DNA pool. In their techniques, the strings in a codeword are appended with an indexing prefix, a solution already incurs $\Theta(M)$ redundant bits, or $\log(e)M - o(1)$ redundancy [62, Remark 1], and will be shown to be strictly sub-optimal in our case. Such index based construction was also considered in [61] and [83], which attempted to correct errors in the index.

In a different setting, where substitution errors occur probabilistically and the decoding error fades with M and L, using indexing prefix was proved asymptotically optimal in terms of coding rate [85]. The indexing prefix technique was also studied in an adversarial setting under substitution errors [61, 83], where codes correcting errors in the indexing prefix were proposed.

The recent work of [62] addressed this model under a bounded number of substitutions, deletions, and insertions per string. When discussing substitutions only, [62] suggested a code construction for $k = 1$ with $2L + 1$ bits of redundancy. Furthermore, by using a reduction to constant Hamming weight vector-codes, it is shown that there exists a code that can correct e errors in each one of the M sequences with redundancy $Me \log(L + 1)$.

The work of [89] studied a more generalized model, where in addition to substitution errors, insertions/deletions of strings were considered. A distance called sequence-subset distance was defined, and upper bounds and constructions were presented based on the sequence-subset distance. When considering only substitution errors, the upper bounds deal with cases where the number of errors is at least a fraction of L.

The work of [54] addressed a similar model, where *multisets* are received at the decoder, rather than sets. In addition, errors in the stored strings are not seen in a fine-grained manner. That is, any set of errors in an individual string is counted as a single error, regardless of how many substitutions, insertions, or deletions it contains. As a result, the specific structure of $\{0, 1\}^L$ is immaterial, and the problem reduces to decoding *histograms* over an alphabet of a certain size.

The specialized reader might suggest the use of *fountain codes*, such as the LT [68] codes or Raptor [84] codes. However, we stress that these solutions rely on randomness at much higher redundancy rates, whereas this work aims for a deterministic and rigorous solution at redundancy which is close to optimal.

Finally, we also mention the *permutation channel* [55, 58, 96], which is similar to our setting, and yet it is farther away in spirit than the aforementioned works. In that channel, a vector over a certain alphabet is transmitted, and its symbols are received at the decoder under a certain permutation. If no restriction is applied over the possible permutations, than this channel reduces to *multiset decoding*, as in [54]. This channel is applicable in networks in which different packets are routed along different paths of varying lengths, and are obtained in an unordered and possibly erroneous form at the decoder. Yet, this line of works is less relevant to ours, and to DNA storage in general, since the specific error pattern in each "symbol" (which corresponds to a string in $\{0, 1\}^L$ in our case) is not addressed, and perfect knowledge of the number of appearances of each "symbol" is assumed.

5.4 Bounds

In this section we use sphere packing arguments in order to establish an existence result of codes with low redundancy, and a lower bound on the redundancy of any k-substitution code. The latter bound demonstrates the asymptotic optimality of the construction in Sec. 5.5 for $k = 1$, up to constants. Our techniques rely on upper and lower bounds on the size of the ball $\mathcal{B}_k$ (Definition 5.2.1), which are given below. However, since our measure for distance is not a metric, extra care is needed when applying sphere-packing arguments. We begin with the existential upper bound in Subsection 5.4, continue to provide a lower bound for $k = 1$ in Subsection 5.4, and extend this bound to larger values of k in Subsection 5.4.

Existential upper bound

In this subsection, let k, M, and L be positive integers such that $k \leq ML$ and $M \leq 2^L$. The subsequent series of lemmas will eventually lead to the following upper bound.

Theorem 5.4.1. *There exists a k-substitution code $C \subseteq \binom{\{0,1\}^L}{M}$ such that $r(C) \leq 2k\log(ML) + 2$.*

We begin with a simple upper bound on the size of the ball $\mathcal{B}_k$.

Lemma 5.4.1. *For every word $W = \{\mathbf{x}_i\}_{i=1}^M \in \binom{\{0,1\}^L}{M}$ and every positive integer $k \leq ML$, we have that $|\mathcal{B}_k(W)| \leq \sum_{\ell=0}^{k}\binom{ML}{\ell}$.*

Proof. Every word in $\mathcal{B}_k(W)$ is obtained by flipping the bits in $\mathbf{x}_i$ that are indexed by some $J_i \subseteq [L]$, for every $i \in [M]$, where $\sum_{i=1}^M |J_i| \leq k$. Clearly, there are at most $\sum_{\ell=0}^{k}\binom{ML}{\ell}$ ways to choose the index sets $\{J_i\}_{i=1}^M$. □

For $W \in \binom{\{0,1\}^L}{\leq M}$ let $\mathcal{R}_k(W)$ be the set of all words $U \in \binom{\{0,1\}^L}{M}$ such that $W \in \mathcal{B}_k(U)$. That is, for a channel output W, the set $\mathcal{R}_k(W)$ contains all potential codewords U whose transmission through the channel can result in W, given that at most k substitutions occur. Further, for $W \in \binom{\{0,1\}^L}{M}$ define the *confusable set* of W as $\mathcal{G}_k(W) \triangleq \cup_{W' \in \mathcal{B}_k(W)}\mathcal{R}_k(W')$. It is readily seen that the words in the confusable set $\mathcal{G}_k(W)$ of a word W cannot reside in the same k-substitution code as W, and therefore we have the following lemma.

Lemma 5.4.2. *For every k, M, and L such that $k \leq ML$ and $M \leq 2^L$ there exists a k-substitution code C such that*

$$|C| \geq \left\lfloor \frac{\binom{2^L}{M}}{D} \right\rfloor, \textit{ where}$$

$$D \triangleq \max_{W \in \binom{\{0,1\}^L}{M}} |\mathcal{G}_k(W)|.$$

Proof. Initialize a list $\mathcal{P} = \binom{\{0,1\}^L}{M}$, and repeat the following process.

1. Choose $W \in \mathcal{P}$.
2. Remove $\mathcal{G}_k(W)$ from $\mathcal{P}$.

Clearly, the resulting code $\mathcal{C}$ is of the aforementioned size. It remains to show that $\mathcal{C}$ corrects k substitutions, i.e., that $\mathcal{B}_k(C) \cap \mathcal{B}_k(C') = \varnothing$ for every distinct $C, C' \in \mathcal{C}$. Recall Definition 5.2.1 that $\mathcal{B}_k(C)$ can be obtained by substitution any k symbols in strings of word C.

Assume for contradiction that there exist distinct $C, C' \in \mathcal{C}$ and $V \in \binom{\{0,1\}^L}{\leq M}$ such that $V \in \mathcal{B}_k(C) \cap \mathcal{B}_k(C')$, and w.l.o.g assume that C was chosen earlier than C' in the above process. Since $V \in \mathcal{B}_k(C)$, it follows that $\mathcal{R}_k(V) \subseteq \mathcal{G}_k(C)$. In addition, since $V \in \mathcal{B}_k(C')$, it follows that $C' \in \mathcal{R}_k(V)$. Therefore, a contradiction is obtained, since C' is in $\mathcal{G}_k(C)$, that was removed from the list $\mathcal{P}$ when C was chosen. $\square$

Lemma 5.4.3. *For an nonnegative integer $T \leq k$ and $W \in \binom{\{0,1\}^L}{M-T}$ we have that $|\mathcal{R}_k(W)| \leq 2(2ML)^k$.*

Proof. Denote $W = \{\mathbf{y}_1, \ldots, \mathbf{y}_{M-T}\}$ and let $U \in \mathcal{R}_k(W)$. Notice that by the definition of $\mathcal{R}_k(W)$, there exists a k-substitution operation which turns U to W. Therefore, every $\mathbf{y}_i$ in W is a result of a certain nonnegative number of substitutions in one or more strings in U. Hence, we denote by $\mathbf{z}_1^1, \ldots, \mathbf{z}_{i_1}^1$ the strings in U that resulted in $\mathbf{y}_1$ after the k-substitution operation, we denote by $\mathbf{z}_1^2, \ldots, \mathbf{z}_{i_2}^2$ the strings which resulted in $\mathbf{y}_2$, and so on, up to $\mathbf{z}_1^{M-T}, \ldots, \mathbf{z}_{i_{M-T}}^{M-T}$, which resulted in $\mathbf{y}_{M-T}$. Therefore, since $U = \cup_{j=1}^{M-T}\{\mathbf{z}_1^j, \ldots, \mathbf{z}_{i_j}^j\}$, it follows that there exists a set $\mathcal{L} \subseteq [M] \times [L]$, of size at most k, such that

$$\begin{pmatrix} \mathbf{z}_1^1 \\ \vdots \\ \mathbf{z}_{i_1}^1 \\ \mathbf{z}_1^2 \\ \vdots \\ \mathbf{z}_{i_{M-T-1}}^{M-T-1} \\ \mathbf{z}_1^{M-T} \\ \vdots \\ \mathbf{z}_{i_{M-T}}^{M-T} \end{pmatrix} = \begin{pmatrix} \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_{M-T-1} \\ \mathbf{y}_{M-T} \\ \vdots \\ \mathbf{y}_{M-T} \end{pmatrix}^{(\mathcal{L})}, \tag{5.1}$$

where $(\cdot)^{(\mathcal{L})}$ is a matrix operator, which corresponds to flipping the bits that are indexed by $\mathcal{L}$ in the matrix on which it operates. In what follows, we bound the number of ways to choose $\mathcal{L}$, which will consequently provide a bound on $|\mathcal{R}_k(W)|$. Note that the number of ways to choose $\mathcal{L}$ is summed over all possible tuples $(i_1, \ldots, i_{M-T})$.

First, define $\mathcal{P} = \{p : i_p > 1\}$, and denote $P \triangleq |\mathcal{P}|$. Therefore, since $\sum_{j=1}^{M-T} i_j = M$, it follows that

$$\begin{aligned} \sum_{p\in\mathcal{P}} i_p = &\sum_{j=1}^{M-T} i_j - \sum_{j\notin\mathcal{P}} i_j \\ =&M - (M - T - P) \\ =&T + P. \end{aligned} \tag{5.2}$$

Second, notice that for every $p \in \mathcal{P}$, the set $\{\mathbf{z}_1^p, \ldots, \mathbf{z}_{i_p}^p\}$ contains i_p different strings. Hence, since after the k-substitution operation they are all equal to $\mathbf{y}_p$, it follows that at least $i_p - 1$ of them must undergo at least one substitution. Clearly, there are $\binom{i_p}{i_p-1} = i_p$ different ways to choose who will these $i_p - 1$ strings be, and additional L^{i_p-1} different ways to determine the locations of the substitutions, and therefore $i_p \cdot L^{i_p-1}$ ways to choose these $i_p - 1$ substitutions.

Third, notice that

$$\begin{aligned} k - \sum_{p\in\mathcal{P}} (i_p - 1) =&k - \sum_{p\in\mathcal{P}} i_p + P \\ \overset{(5.2)}{=}&k - T, \end{aligned} \tag{5.3}$$

and hence, there are at most $k - T$ remaining positions to be chosen to $\mathcal{L}$, after choosing the $i_p - 1$ positions for every $p \in \mathcal{P}$ as described above.

Now, let $\mathcal{I}$ be the set of all tuples $(i_1, \ldots, i_{M-T})$ of positive integers that sum to M (whose size is $\binom{M-1}{M-T-1}$ by the famous *stars and bars* theorem). Let $N : \mathcal{I} \to \mathbb{N}$ be a function which maps $(i_1, \ldots, i_{M-T}) \in \mathcal{I}$ to the number of different $U \in \mathcal{R}_k(W)$ for which there exist $\mathcal{L} \subseteq [M] \times [L]$ of size at most k such that (5.1) is satisfied. Since this quantity is at most the number of ways to choose a suitable $\mathcal{L}$, the above arguments demonstrate that

$$N(i_1, \ldots, i_{M-T}) \leq \binom{ML}{k-T} \prod_{p\in\mathcal{P}} i_p L^{i_p - 1}.$$

Then, we have

$$\begin{aligned}
|\mathcal{R}_k(W)| &\leq \sum_{\mathcal{I}} N(i_1, \ldots, i_{M-T}) \\
&\leq \sum_{\mathcal{I}} \binom{ML}{k-T} \prod_{p\in\mathcal{P}} i_p L^{i_p-1} \\
&\leq \sum_{\mathcal{I}} (ML)^{k-T} L^{\sum_p (i_p - 1)} \prod_{p\in\mathcal{P}} i_p \\
&\overset{(5.3)}{\leq} \sum_{\mathcal{I}} (ML)^{k-T} L^T \prod_{p\in\mathcal{P}} i_p. \qquad (5.4)
\end{aligned}$$

Since the geometric mean of positive numbers is always less than the arithmetic one, we have $\left(\prod_{p\in\mathcal{P}} i_p\right)^{1/P} \leq \frac{1}{P}\sum_{p\in\mathcal{P}} i_p$, and hence,

$$\begin{aligned}
(5.4) &\leq \sum_{\mathcal{I}} (ML)^{k-T} L^T \left(\tfrac{\sum_p i_p}{P}\right)^P \\
&\overset{(5.2)}{\leq} \sum_{\mathcal{I}} (ML)^{k-T} L^T ((T+P)/P)^P \\
&\leq \binom{M-1}{M-T-1} (ML)^{k-T} L^T ((T+P)/P)^P \\
&\leq \binom{M}{T} (ML)^{k-T} L^T ((T+P)/P)^P \\
&\leq (ML)^{k-T} (ML)^T ((T+P)/P)^P \\
&\leq (ML)^k ((T+P)/P)^P \\
&\overset{(a)}{\leq} (ML)^k 2^T \\
&\leq (2ML)^k, \qquad (5.5)
\end{aligned}$$

where (a) will be proved in Appendix 5.9. □

Proof. (of Theorem 5.4.1) It follows from Lemma 5.4.1, Lemma 5.4.3, and from the definition of D that

$$\begin{aligned} D &\le \max_{W\in\binom{\{0,1\}^L}{M}} |\mathcal{B}_k(W)| \cdot \max_{W\in\binom{\{0,1\}^L}{M}} |\mathcal{R}_k(W)| \\ &\le \left(\sum_{\ell=0}^{k}\binom{ML}{\ell}\right)(2ML)^k. \end{aligned}$$

Therefore, the code C that is constructed in Lemma 5.4.2 satisfies

$$\begin{aligned} r(C) &\le \log\binom{2^L}{M} - \log|C| \\ &\le \log\left(\left(\sum_{\ell=0}^{k}\binom{ML}{\ell}\right)(2ML)^k\right) \\ &= \log\left(\sum_{\ell=0}^{k}\binom{ML}{\ell}\right) + k(\log(ML)+1) \\ &\le \log\left(k\binom{ML}{k}\right) + k(\log(ML)+1) \\ &\le \left(\log(k) - \log(k!) + \log(ML^k)\right) \\ &+ k(\log(ML)+1) \\ &\le 2k\log(ML) + 2. \end{aligned}$$

□

Lower bound for a single substitution code

Notice that the bound in Lemma 5.4.1 is tight, e.g., in cases where $d_H(\mathbf{x}_i,\mathbf{x}_j) \ge 2k+1$ for all distinct $i,j\in[M]$. This might occur only if M is less than the maximum size of a k-substitution correcting vector-code, i.e., when $M \le 2^L/(\sum_{i=0}^{k}\binom{L}{i})$ [79, Sec. 4.2]. When the minimum Hamming distance between the strings in a codeword is not large enough, different substitution errors might result in identical words, and the size of the ball is smaller than the given upper bound. This is illustrated in the following example.

Example 5.4.1. *For $L=3$ and $M=2$, consider the word $W=\{\underline{0}10,\mathbf{0}1\underline{\mathbf{1}}\}$. By flipping either the two underlined symbols, or the two bold symbols, the word $W'=\{010,110\}$ is obtained. Hence, different substitution operations might result in identical words. As a result, the size of the ball*

$$\begin{aligned} \mathcal{B}_2(W) = \{&\{010,011\},\{110,011\},\{000,011\},\{011\}, \\ &\{010,111\},\{010,001\},\{010\},\{100,011\}, \end{aligned}$$

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

is 17*, which is smaller than the upper bound of the* $\mathcal{B}_2(W)$ *ball size* $\binom{ML}{0}+\binom{ML}{1}+\binom{ML}{2}=22$.

However, in some cases it is possible to bound the size of $\mathcal{B}_k$ from below by using tools from Fourier analysis of Boolean functions. In the following it is assumed that that $k=1$. A word $W \in \binom{\{0,1\}^L}{M}$ corresponds to a Boolean function $f_W : \{\pm 1\}^L \to \{\pm 1\}$ as follows. For $\mathbf{x} \in \{0,1\}^L$ let $\overline{\mathbf{x}} \in \{\pm 1\}^L$ be the vector which is obtained from $\mathbf{x}$ be replacing every 0 by 1 and every 1 by -1. Then, we define $f_W(\overline{\mathbf{x}}) = -1$ if $\mathbf{x} \in W$, and 1 otherwise. Considering the set $\{\pm 1\}^L$ as the *hypercube graph*[3], the *boundary* ∂f_W of f_W is the set of all edges $\{\mathbf{x}_1,\mathbf{x}_2\} \in \binom{\{\pm 1\}^L}{2}$ in this graph such that $f_W(\mathbf{x}_1) \neq f_W(\mathbf{x}_2)$.

Lemma 5.4.4. *For every word* $W \in \binom{\{0,1\}^L}{M}$ *we have that* $|\mathcal{B}_1(W)| \geq |\partial f_W| + 1$.

Proof. Let $W = \{\mathbf{x}_1,\ldots,\mathbf{x}_M\}$ be a word. Every edge $e = \{\mathbf{x}_i,\mathbf{x}'\}$ on the boundary of f_W corresponds to a substitution operation in $\mathbf{x}_i$ from W that results in a word $W_e = \{\mathbf{x}_1,\ldots,\mathbf{x}_{i-1},\mathbf{x}',\mathbf{x}_{i+1},\ldots,\mathbf{x}_M\} \in (\mathcal{B}_1(W)\setminus\{W\}) \cap \binom{\{0,1\}^L}{M}$. To show that every edge e on the boundary corresponds to a *unique* word W_e in $\mathcal{B}_1(W)$, assume for contradiction that $W_e = W_{e'}$ for two distinct edges $e = \{\overline{\mathbf{x}}_1,\overline{\mathbf{x}}_2\}$ and $e' = \{\overline{\mathbf{y}}_1,\overline{\mathbf{y}}_2\}$, where $\mathbf{x}_1,\mathbf{y}_1 \in W$ and $\mathbf{x}_2,\mathbf{y}_2 \notin W$. Since both W_e and $W_{e'}$ contain precisely one element which is not in W, and are missing one element which is in W, it follows that $\mathbf{x}_1 = \mathbf{y}_1$ and $\mathbf{x}_2 = \mathbf{y}_2$, a contradiction. Therefore, there exists an injective mapping between the boundary of f_W and $\mathcal{B}_1(W)\setminus\{W\}$, and the claim follows. □

Notice that the bound in Lemma 5.4.4 is tight, e.g., in cases where the minimum Hamming distance between the strings of W is at least 2. This implies the tightness of the bound which is given below in these cases. Having established the connection between $\mathcal{B}_1(W)$ and the boundary of f_W, the following Fourier analytic claim will aid in proving a lower bound. Let the *total influence* of f_W be $I(f_W) \triangleq \sum_{i=1}^{L} \Pr_{\mathbf{x}}(f_W(\mathbf{x}) \neq f_W(\mathbf{x}^{\oplus i}))$, where $\mathbf{x}^{\oplus i}$ is obtained from $\mathbf{x}$ by changing the sign of the i-th entry, and $\mathbf{x}$ is chosen uniformly at random.

[3]The nodes of the hypercube graph of dimension L are identified by $\{\pm 1\}^L$, and every two nodes are connected if and only if the Hamming distance between them is 1.

Lemma 5.4.5. *[72, Theorem 2.39] For every function $f : \{\pm 1\}^L \to \mathbb{R}$, we have that $I(f) \geq 2\alpha \log(1/\alpha)$, where $\alpha = \alpha(f) \triangleq \min\{\Pr_{\mathbf{x}}(f(\mathbf{x}) = 1), \Pr_{\mathbf{x}}(f(\mathbf{x}) = -1)\}$, and $\mathbf{x} \in \{\pm 1\}^L$ is chosen uniformly at random.*

Lemma 5.4.6. *Let L and M be integers such that $M \leq 2^{(1-\epsilon)L}$ and $L > \frac{1}{\epsilon}$ for some $0 < \epsilon < 1$. For every word $W \in \binom{\{0,1\}^L}{M}$ we have that $|\partial f_W| \geq \epsilon ML$.*

Proof. Since $M \leq 2^{(1-\epsilon)L}$ and $\alpha = \alpha(f_W) = \min\{(2^L - M)/2^L, M/2^L\}$, it follows that $\alpha = M/2^L$ whenever $L > \frac{1}{\epsilon}$, which holds for every non-constant L. In addition, since $\Pr_{\mathbf{x}}(f_W(\overline{\mathbf{x}}) \neq f_W(\overline{\mathbf{x}}^{\oplus i}))$ equals the fraction of dimension i edges that lie on the boundary of f_W ([**O'Donnell**]), Lemma 5.4.4 implies that

$$I(f_W) = \frac{|\partial f_W|}{2^{L-1}}.$$

Therefore, since $M \leq 2^{(1-\epsilon)L}$ and from Lemma 5.4.5 we have that $|\partial f_W| = 2^{L-1} I(f_W) \geq 2^L \alpha \log(1/\alpha) = M \log(2^L/M) \geq \epsilon ML$. □

Corollary 5.4.1. *For integers L and M and a constant $0 < \epsilon < 1$ such that $M \leq 2^{(1-\epsilon)L}$ and $L > \frac{1}{\epsilon}$, any 1-substitution code $C \subseteq \binom{\{0,1\}^L}{M}$ satisfies that $r(C) \geq \log(ML) - \log 1/\epsilon$.*

Proof. According to Lemma 5.4.4 and Lemma 5.4.6, every codeword of every C excludes at least ϵML other words from belonging to C. Hence, we have that $|C| \leq \binom{2^L}{M}/\epsilon ML$, and by the definition of redundancy, it follows that

$$\begin{aligned} r(C) &= \log \binom{2^L}{M} - \log(|C|) \\ &\geq \log(\epsilon ML) \\ &= \log(ML) - \log 1/\epsilon. \end{aligned}$$ □

Remark 5.4.1. *A similar lower bound was presented in [62], where it was shown that for a code correcting s loss of strings, q substitution errors in each of t strings, the redundancy is lower bounded by*

$$sL + t \log M + tq \log L - \log(s!t!q!^t) + o(1),$$

when $M = 2^{\beta L}$ for some $0 < \beta < 1$. Taking $s = 0$ and $q = t = 1$ results in the lower bound $\log(ML) + o(1)$, which is order-wise the same as, and stronger by a constant $\log 1/\epsilon$ than the lower bound $\log(ML) - \log 1/\epsilon$ in Corollary 5.4.1. Compared to the bound in [62], the bound in Corollary 5.4.1 does not require M to be exponential in L, and thus applies to broader parameter range for M and L.

Lower bound for more than one substitution

Similar techniques to the ones in Subsection 5.4 can be used to obtain a lower bound for larger values of k. Specifically, we have the following theorem.

Theorem 5.4.2. *For integers L, M, k, and positive constants $\epsilon, c < 1$ such that $M \leq 2^{(1-\epsilon)L}$, $L > \frac{1}{\epsilon}$, and $k \leq c\epsilon\sqrt{M}$, a k-substitution code $C \subseteq \binom{\{0,1\}^L}{M}$ satisfies that $r(C) \geq k(\log(ML) - 2\log(k)) - O(1)$.*

To prove this theorem, it is shown that certain special k-subsets of ∂f_W correspond to words in $\mathcal{B}_k(W)$, and by bounding the number of these special subsets from below, the lower bound is attained. A subset of k boundary edges is called *special*, if it does not contain two edges that intersect on a node (i.e., a string) in W. Formally, a subset $\mathcal{S} \subseteq \partial f_W$ is special if $|\mathcal{S}| = k$, and for every $\{\mathbf{x}_1, \mathbf{y}_1\}, \{\mathbf{x}_2, \mathbf{y}_2\} \in \mathcal{S}$ with $f_W(\overline{\mathbf{x}}_1) = f_W(\overline{\mathbf{x}}_2) = -1$ and $f_W(\overline{\mathbf{y}}_1) = f_W(\overline{\mathbf{y}}_2) = 1$ we have that $\mathbf{x}_1 \neq \mathbf{x}_2$. We begin by showing how special sets are relevant to proving Theorem 5.4.2.

Lemma 5.4.7. *For every word $W \in \binom{\{0,1\}^L}{M}$ we have that*

$$|\mathcal{B}_k(W)| \geq \frac{|\{\mathcal{S} \subseteq \partial f_W | \mathcal{S} \text{ is special}\}|}{k^k}.$$

Proof. It is shown that every special set corresponds to a word in $\mathcal{B}_k(W)$, and at most k^k different special sets can correspond to the same word (namely, there exists a mapping from the family of special sets to $\mathcal{B}_k(W)$, which is at most k^k to 1). Let $\mathcal{S} = \{\{\mathbf{x}_i, \mathbf{y}_i\}\}_{i=1}^k$ be special, where $f_W(\overline{\mathbf{x}}_i) = -1$ and $f_W(\overline{\mathbf{y}}_i) = 1$ for every $i \in [k]$. Let $W_\mathcal{S} \in \binom{\{0,1\}^L}{\leq M}$ be obtained from W by removing the $\mathbf{x}_i$'s and adding the $\mathbf{y}_i$'s, i.e., $W_\mathcal{S} \triangleq (W \setminus \{\mathbf{x}_i\}_{i=1}^k) \cup \{\mathbf{y}_i\}_{i=1}^T$ for some $T \leq k$; notice that there are exactly k distinct $\mathbf{x}_i$'s but at most k distinct $\mathbf{y}_i$'s, since $\mathcal{S}$ is special, and therefore we assume w.l.o.g that $\mathbf{y}_1, \ldots, \mathbf{y}_T$ are the distinct $\mathbf{y}_i$'s. It is readily verified that $W_\mathcal{S} \in \mathcal{B}_k(W)$, since $W_\mathcal{S}$ can be obtained from W by performing k substitution operations in W, each of which corresponds to an edge in $\mathcal{S}$. Moreover, every $\mathcal{S}$ corresponds to a unique surjective function $f_\mathcal{S} : [k] \to [T]$ such that $f_\mathcal{S}(i) = j$ if there exists $j \leq T$ such that $\{\mathbf{x}_i, \mathbf{y}_j\} \in \mathcal{S}$, and hence at most $k^T \leq k^k$ different special sets $\mathcal{S}$ can correspond to the same word in $\mathcal{B}_k(W)$. □

We now turn to prove a lower bound on the number of special sets.

Lemma 5.4.8. *Let L and M be integers such that $M \leq 2^{(1-\epsilon)L}$ and $L > \frac{1}{\epsilon}$ for some $0 < \epsilon < 1$. If there exists a positive constant $c < 1$ such that $k \leq c \cdot \epsilon\sqrt{M}$, then there are at least $(1 - c^2)\binom{|\partial f_W|}{k}$ special sets $\mathcal{S} \subseteq \partial f_W$.*

Proof. Clearly, the number of ways to choose a k-subset of ∂f_W which is *not* special, i.e., contains k distinct edges of ∂f_W but at least two of those are adjacent to the same $\mathbf{x} \in W$, is at most

$$\begin{aligned} &M \cdot \binom{L}{2} \cdot \binom{|\partial f_W|}{k-2} \\ =&M \cdot \binom{L}{2} \cdot \frac{k(k-1)}{(|\partial f_W| - k + 2)(|\partial f_W| - k + 1)} \cdot \binom{|\partial f_W|}{k}. \end{aligned}$$

Observe that the multiplier of $\binom{|\partial f_W|}{k}$ in the above expression can be bounded as follows.

$$\begin{aligned} &M \cdot \binom{L}{2} \cdot \frac{k(k-1)}{(|\partial f_W| - k + 2)(|\partial f_W| - k + 1)} \\ \le &M \cdot \binom{L}{2} \cdot \frac{k(k-1)}{(\epsilon ML - k + 2)(\epsilon ML - k + 1)} \\ \le &M \cdot L^2 \cdot \frac{k^2}{\epsilon^2 M^2 L^2}, \end{aligned}$$

where the former inequality follows since $|\partial f_W| \ge \epsilon ML$ by Lemma 5.4.6; the latter inequality follows since $k \le c\epsilon\sqrt{M}$ implies that $\epsilon ML - k + 2 \ge \epsilon ML - k + 1 \ge \frac{1}{\sqrt{2}} \cdot \epsilon ML$ whenever $\frac{c\sqrt{2}}{\sqrt{2}-1} \le L\sqrt{M}$, which holds for every non-constant M and L. Therefore, since

$$M \cdot L^2 \cdot \frac{k^2}{\epsilon^2 M^2 L^2} = \frac{k^2}{\epsilon^2 M} \le c^2,$$

it follows that these are at least $(1 - c^2) \cdot \binom{|\partial f_W|}{k}$ special subsets in ∂f_W. □

Lemma 5.4.7 and Lemma 5.4.8 readily imply that $|\mathcal{B}_k(W)| \ge \frac{(1-c^2)}{k^k}\binom{\epsilon ML}{k}$ for every $W \in \binom{\{0,1\}^L}{M}$, from which we can prove Theorem 5.4.2.

Proof. (of Theorem 5.4.2) Clearly, no two k-balls around codewords in C can intersect, and therefore we must have $|C| \le \binom{2^L}{M} / \min_{W \in C} |\mathcal{B}_k(W)|$. Therefore,

$$\begin{aligned} r(C) &= \log \binom{2^L}{M} - \log |C| \\ &\ge \log \left(\frac{(1-c^2)}{k^k} \binom{\epsilon ML}{k} \right) \\ &= \log \binom{\epsilon ML}{k} - k \log(k) - O(1) \\ &\ge \log \left(\frac{(\epsilon ML - k + 1)^k}{k^k} \right) - k \log(k) - O(1) \end{aligned}$$

$$\geq \log\left(\frac{\left(\frac{1}{\sqrt{2}}\epsilon ML\right)^k}{k^k}\right) - k\log(k) - O(1)$$
$$= k\left(\log(\tfrac{\epsilon}{\sqrt{2}}) + \log(ML)\right) - 2k\log(k) - O(1)$$
$$\geq k\log(ML) - 2k\log(k) - O(k). \qquad \square$$

5.5 Codes for a Single Substitution

In this section we present a 1-substitution code construction that applies whenever $M \leq 2^{L/6}$, whose redundancy is $3\log ML + 3\log M + O(1)$. For simplicity of illustration, we restrict our attention to values of M and L such that $\lceil\log ML\rceil + \lceil\log M\rceil \leq M$. In the remaining values, a similar construction of comparable redundancy exists.

Theorem 5.5.1. *For $D = [\binom{2^{L/3-1}}{M}^3 \cdot (M!)^2 \cdot 2^{3M-3\log ML-3\log M-6}]$, there exist an encoding function $E : D \to \binom{\{0,1\}^L}{M}$ whose image is a single substitution correcting code.*

The idea behind Theorem 5.5.1 is to concatenate the strings in a codeword $C = \{\mathbf{x}_i\}_{i=1}^M$ in a certain order, so that classic 1-substitution error correction techniques can be applied over the concatenated string. Since a substitution error may affect any particular order of the $\mathbf{x}_i$'s, we consider the lexicographic orders of several different parts of the $\mathbf{x}_i$'s, instead of the lexicographic order of the whole strings. Specifically, we partition the $\mathbf{x}_i$'s to three parts, and place distinct strings in each of them. Since a substitution operation can scramble the order in at most one part, the correct order will be inferred by a majority vote, so that classic substitution error correction can be applied.

Consider a message $d \in D$ as a tuple $d = (d_1, \ldots, d_6)$, where

$$d_1 \in [\binom{2^{L/3-1}}{M}],$$

$$d_3, d_5 \in [\binom{2^{L/3-1}}{M}M!],$$

and

$$d_2, d_4, d_6 \in [2^{M-\log ML-\log M-2}].$$

Apply the functions F_{com}, F_{perm}, and F (see Sec. 7.1.1) to obtain

$$F_{com}(d_1) = \{\mathbf{a}_1, \ldots, \mathbf{a}_M\},$$

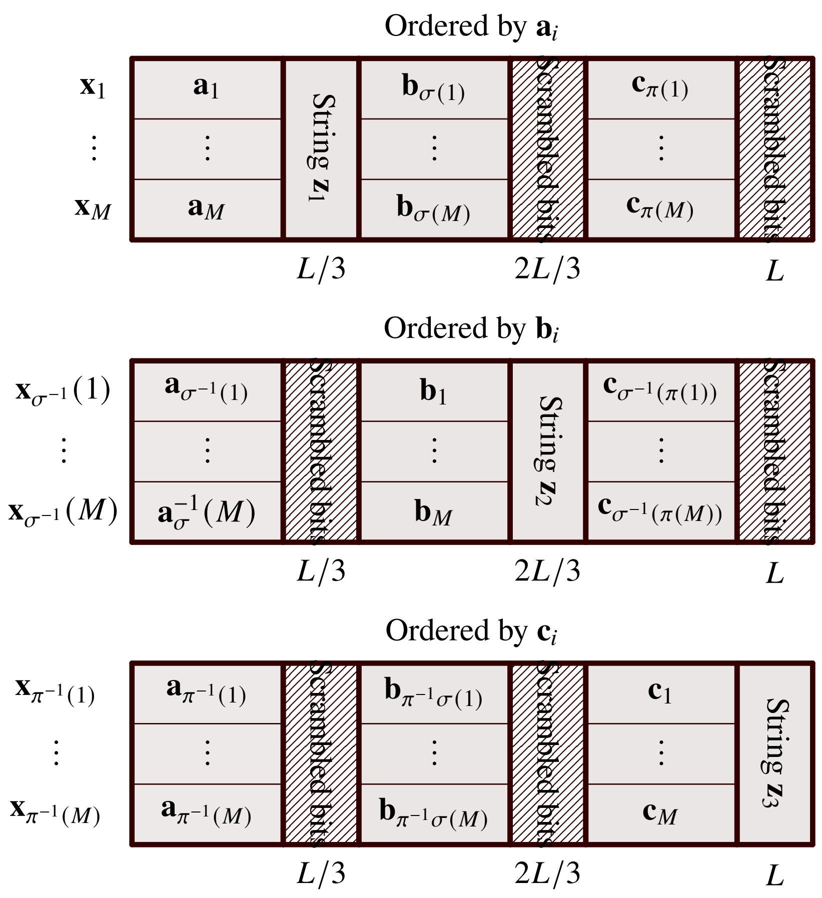
Ordered by $\mathbf{a}_i$
$\mathbf{x}_1$
$\mathbf{x}_M$
$\mathbf{a}_1$
$\mathbf{a}_M$
String $\mathbf{z}_1$
$\mathbf{b}_{\sigma(1)}$
$\mathbf{b}_{\sigma(M)}$
Scrambled bits
$\mathbf{c}_{\pi(1)}$
$\mathbf{c}_{\pi(M)}$
Scrambled bits
$L/3$
$2L/3$
L
Ordered by $\mathbf{b}_i$
$\mathbf{x}_{\sigma^{-1}}(1)$
$\mathbf{x}_{\sigma^{-1}}(M)$
$\mathbf{a}_{\sigma^{-1}(1)}$
$\mathbf{a}_{\sigma}^{-1}(M)$
Scrambled bits
$\mathbf{b}_1$
$\mathbf{b}_M$
String $\mathbf{z}_2$
$\mathbf{c}_{\sigma^{-1}(\pi(1))}$
$\mathbf{c}_{\sigma^{-1}(\pi(M))}$
Scrambled bits
$L/3$
$2L/3$
L
Ordered by $\mathbf{c}_i$
$\mathbf{x}_{\pi^{-1}(1)}$
$\mathbf{x}_{\pi^{-1}(M)}$
$\mathbf{a}_{\pi^{-1}(1)}$
$\mathbf{a}_{\pi^{-1}(M)}$
Scrambled bits
$\mathbf{b}_{\pi^{-1}\sigma(1)}$
$\mathbf{b}_{\pi^{-1}\sigma(M)}$
Scrambled bits
$\mathbf{c}_1$
$\mathbf{c}_M$
String $\mathbf{z}_3$
$L/3$
$2L/3$
L

Figure 5.1: Illustration of single-substitution correcting codes over unordered sets.

$$\begin{aligned} F(d_3) &= (\{\mathbf{b}_1, \ldots, \mathbf{b}_M\}, \sigma), \\ F(d_5) &= (\{\mathbf{c}_1, \ldots, \mathbf{c}_M\}, \pi), \end{aligned} \tag{5.6}$$

where $\mathbf{a}_i, \mathbf{b}_i, \mathbf{c}_i \in \{0,1\}^{L/3-1}$ for every $i \in [M]$, the permutations σ and π are in S_M, and the indexing of $\{\mathbf{a}_i\}_{i=1}^M$, $\{\mathbf{b}_i\}_{i=1}^M$, and $\{\mathbf{c}_i\}_{i=1}^M$ is lexicographic. Further, let $\mathbf{d}_2, \mathbf{d}_4$, and $\mathbf{d}_6$ be the binary strings that correspond to d_2, d_4, and d_6, respectively, and let

$$\begin{aligned} \mathbf{s}_1 = (&\mathbf{a}_1, \ldots, \mathbf{a}_M, \mathbf{b}_{\sigma(1)}, \ldots, \mathbf{b}_{\sigma(M)}, \\ &\mathbf{c}_{\pi(1)}, \ldots, \mathbf{c}_{\pi(M)}), \\ \mathbf{s}_2 = (&\mathbf{a}_{\sigma^{-1}(1)}, \ldots, \mathbf{a}_{\sigma^{-1}(M)}, \mathbf{b}_1, \ldots, \mathbf{b}_M, \\ &\mathbf{c}_{\sigma^{-1}\pi(1)}, \ldots, \mathbf{c}_{\sigma^{-1}\pi(M)}), \text{ and} \\ \mathbf{s}_3 = (&\mathbf{a}_{\pi^{-1}(1)}, \ldots, \mathbf{a}_{\pi^{-1}(M)}, \mathbf{b}_{\pi^{-1}\sigma(1)}, \ldots, \mathbf{b}_{\pi^{-1}\sigma(M)}, \\ &\mathbf{c}_1, \ldots, \mathbf{c}_M). \end{aligned} \tag{5.7}$$

Without loss of generality[4] assume that there exists an integer t for which the length $|\mathbf{s}_i| = (L-3)M = 2^t - t - 1$ for all $i \in [3]$. Then, each $\mathbf{s}_i$ can be encoded by using a *systematic* $[2^t-1, 2^t-t-1]_2$ Hamming code, by introducing t redundant bits. That is, the encoding function is of the form $\mathbf{s}_i \mapsto (\mathbf{s}_i, E_H(\mathbf{s}_i))$, where $E_H(\mathbf{s}_i)$ are the t redundant bits, and $t \leq \lceil \log(ML) \rceil$. Similarly, we assume that there exists an integer h for which the length $|\mathbf{d}_i| = 2^h - h - 1$ for $i \in \{2,4,6\}$, and let $E_H(\mathbf{d}_i)$ be the corresponding h bits of redundancy, that result from encoding $\mathbf{d}_i$ by using a $[2^h-1, 2^h-h-1]$ Hamming code. By the properties of a Hamming code, and by the definition of h, we have that $h \leq \lceil \log(M) \rceil$.

The data $d \in D$ is mapped to a codeword $C = \{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$ as follows, and the reader is encouraged to refer to Figure 5.1 for clarifications. First, we place $\{\mathbf{a}_i\}_{i=1}^M$, $\{\mathbf{b}_i\}_{i=1}^M$, and $\{\mathbf{c}_i\}_{i=1}^M$ in the different thirds of the $\mathbf{x}_i$'s, sorted by σ and π. That is, denoting $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,L})$, we define

$$\begin{aligned}(x_{i,1}, \ldots, x_{i,L/3-1}) &= \mathbf{a}_i,\\ (x_{i,L/3+1}, \ldots, x_{i,2L/3-1}) &= \mathbf{b}_{\sigma(i)}, \text{ and}\\ (x_{i,2L/3+1}, \ldots, x_{i,L-1}) &= \mathbf{c}_{\pi(i)}.\end{aligned} \tag{5.8}$$

The remaining bits $\{x_{i,L/3}\}_{i=1}^M$, $\{x_{i,2L/3}\}_{i=1}^M$, and $\{x_{i,L}\}_{i=1}^M$ are used to accommodate the information bits of $\mathbf{d}_2, \mathbf{d}_4, \mathbf{d}_6$, and the redundancy bits $\{E_H(\mathbf{s}_i)\}_{i=1}^3$ and $\{E_H(\mathbf{d}_i)\}_{i\in\{2,4,6\}}$, in the following manner.

$$x_{i,L/3} = \begin{cases} d_{2,i}, & \\ \qquad \text{if } i \in [M - \lceil \log ML \rceil - \lceil \log M \rceil], & \\ E_H(\mathbf{d}_2)_{i-(M-\lceil \log ML \rceil - \lceil \log M \rceil)}, & \\ \qquad \text{if } i \in [M - \lceil \log ML \rceil - \lceil \log M \rceil + 1, & \\ \qquad M - \lceil \log ML \rceil], h & \\ E_H(\mathbf{s}_1)_{i-(M-\lceil \log ML \rceil)}, & \\ \qquad \text{if } i \in [M - \lceil \log ML \rceil + 1, M], & \end{cases}$$

[4]Every string can be padded with zeros to extend its length to $2^t - t - 1$ for some t. It is readily verified that this operation extends the string by at most a factor of two, and by the properties of the Hamming code, this will increase the number of redundant bits by at most 1.

$$x_{i,2L/3} = \begin{cases} d_{4,\sigma^{-1}(i)}, \\ \qquad \text{if } \sigma^{-1}(i) \in [M - \lceil \log ML \rceil - \lceil \log M \rceil], \\ E_H(\mathbf{d}_4)_{\sigma^{-1}(i)-(M-\lceil \log ML \rceil - \lceil \log M \rceil)}, \\ \qquad \text{if } \sigma^{-1}(i) \in [M - \lceil \log ML \rceil - \lceil \log M \rceil \\ \qquad\ + 1, M - \lceil \log ML \rceil], \\ E_H(\mathbf{s}_2)_{\sigma^{-1}(i)-(M-\lceil \log ML \rceil)}, \\ \qquad \text{if } \sigma^{-1}(i) \in [M - \lceil \log ML \rceil + 1, M], \end{cases}$$

$$x_{i,L} = \begin{cases} d_{6,\pi^{-1}(i)}, \\ \qquad \text{if } \pi^{-1}(i) \in [M - \lceil \log ML \rceil - \lceil \log M \rceil], \\ E_H(\mathbf{d}_6)_{\pi^{-1}(i)-(M-\lceil \log ML \rceil - \lceil \log M \rceil)}, \\ \qquad \text{if } \pi^{-1}(i) \in [M - \lceil \log ML \rceil - \lceil \log M \rceil \\ \qquad\ + 1, M - \lceil \log ML \rceil], \\ E_H(\mathbf{s}_3)_{\pi^{-1}(i)-(M-\lceil \log ML \rceil)}, \\ \qquad \text{if } \pi^{-1}(i) \in [M - \lceil \log ML \rceil + 1, M]. \end{cases} \tag{5.9}$$

That is, if the strings $\{\mathbf{x}_i\}_{i=1}^M$ are sorted according to the content of the bits $(x_{i,1}, \ldots, x_{i,L/3-1}) = \mathbf{a}_i$, then the top $M - \lceil \log ML \rceil - \lceil \log M \rceil$ bits of the $(L/3)$-th column[5] contain $\mathbf{d}_2$, the middle $\lceil \log M \rceil$ bits contain $E_H(\mathbf{d}_2)$, and the bottom $\lceil \log ML \rceil$ bits contain $E_H(\mathbf{s}_1)$. Similarly, if the strings are sorted according to $(x_{i,L/3+1}, \ldots, x_{i,2L/3-1}) = \mathbf{b}_i$, then the top $M - \lceil \log ML \rceil - \lceil \log M \rceil$ bits of the $(2L/3)$-th column contain $\mathbf{d}_4$, the middle $\lceil \log M \rceil$ bits contain $E_H(\mathbf{d}_4)$, and the bottom $\lceil \log ML \rceil$ bits contain $E_H(\mathbf{s}_2)$, and so on. Equations (5.8) and (5.9) conclude the encoding function E of Theorem 5.5.1. It can be readily verified that E is injective since different messages result in either different $(\{\mathbf{a}_i\}_{i=1}^M, \{\mathbf{b}_i\}_{i=1}^M, \{\mathbf{c}_i\}_{i=1}^M)$ or the same $(\{\mathbf{a}_i\}_{i=1}^M, \{\mathbf{b}_i\}_{i=1}^M, \{\mathbf{c}_i\}_{i=1}^M)$ with different $(\mathbf{d}_2, \mathbf{d}_4, \mathbf{d}_6)$. In either case, the resulting codewords $\{\mathbf{x}_i\}_{i=1}^M$ of the two messages are different.

To verify that the image of E is a 1-substitution code, observe first that since $\{\mathbf{a}_i\}_{i=1}^M$, $\{\mathbf{b}_i\}_{i=1}^M$, and $\{\mathbf{c}_i\}_{i=1}^M$ are *sets*, it follows that any two strings in the same set are distinct. Hence, according to (5.8), it follows that $d_H(\mathbf{x}_i, \mathbf{x}_j) \geq 3$ for every distinct i and j in $[M]$. Therefore, no 1-substitution error can cause one $\mathbf{x}_i$ to be equal to another, and consequently, the result of a 1-substitution error is always in $\binom{\{0,1\}^L}{M}$.

[5]Sorting the strings $\{\mathbf{x}_i\}_{i=1}^M$ by any ordering method provides a matrix in a natural way, and can consider columns in this matrix.

In what follows a decoding algorithm is presented, whose input is a codeword that was distorted by at most a single substitution, and its output is d. The algorithm is summarized in Algorithm 3.

Algorithm 3: Decoding

```
Input: A word $C' \in \mathcal{B}_1(C)$ for some codeword $C$.
Output: The message $d$ encoded as $C$.
Sort and index the strings in $C' = \{\mathbf{x}'_1, \ldots, \mathbf{x}'_M\}$ lexicographically;
Compute the strings $\hat{\mathbf{a}}_i$, $\hat{\mathbf{b}}_i$, and $\hat{\mathbf{c}}_i$ for $i \in [M]$, according to (5.10);
Compute the strings $\mathbf{s}'_1$, $\mathbf{s}'_2$, and $\mathbf{s}'_3$ according to (5.12);
Compute the strings $E_H(\mathbf{s}_1)'$, $E_H(\mathbf{s}_2)'$, and $E_H(\mathbf{s}_3)'$ according to (5.11);
Use Hamming decoder to decode $(\mathbf{s}'_i, E_H(\mathbf{s}_i))$ and obtain $\mathbf{s}_i$ for $i \in [3]$;
According to Lemma 5.5.1, we can apply majority vote on the recovered $\{\mathbf{s}_i\}_{i=1}^3$
  to obtain the correct strings $\{\mathbf{a}_i\}_{i=1}^M$, $\{\mathbf{b}_i\}_{i=1}^M$, and $\{\mathbf{c}_i\}_{i=1}^M$, and the
  permutations $\sigma$ and $\pi$. Then determine $d_1, d_3, d_5$ using combinatorial
  map (5.6);
Compute $(\mathbf{d}'_i, E_H(\mathbf{d}_i)')$ $i \in \{2, 4, 6\}$ according to (5.11) and use Hamming
  decoder to decode $(\mathbf{d}'_i, E_H(\mathbf{d}_i)')$ and obtain $\mathbf{d}_i$ for $i \in \{2, 4, 6\}$;
Output $d = (d_1, d_2, d_3, d_4, d_5, d_6)$.
```

Upon receiving a word $C' = \{\mathbf{x}'_1, \ldots, \mathbf{x}'_M\} \in \mathcal{B}_1(C)$ for some codeword C (once again, the indexing of the elements of C' is lexicographic), we define

$$\begin{aligned}
\hat{\mathbf{a}}_i &= (x'_{i,1}, \ldots, x'_{i,L/3-1}) \\
\hat{\mathbf{b}}_i &= (x'_{\tau^{-1}(i),L/3+1}, \ldots, x'_{\tau^{-1}(i),2L/3-1}) \\
\hat{\mathbf{c}}_i &= (x'_{\rho^{-1}(i),2L/3+1}, \ldots, x'_{\rho^{-1}(i),L-1}),
\end{aligned} \tag{5.10}$$

where τ is the permutation by which $\{\mathbf{x}'_i\}_{i=1}^M$ are sorted according to their $L/3 + 1, \ldots, 2L/3 - 1$ entries, and ρ is the permutation by which they are sorted according to their $2L/3 + 1, \ldots, L - 1$ entries (we emphasize that τ and ρ are unrelated to the original π and σ, and those will be decoded later). Further, when ordering $\{\mathbf{x}'_i\}_{i=1}^M$ by either the lexicographic ordering, by τ, or by ρ, we obtain *candidates* for each one of $\mathbf{d}_2$, $\mathbf{d}_4$, $\mathbf{d}_6$, $E_H(\mathbf{d}_2)$, $E_H(\mathbf{d}_4)$, $E_H(\mathbf{d}_6)$, $E_H(\mathbf{s}_1)$, $E_H(\mathbf{s}_2)$, and $E_H(\mathbf{s}_3)$, that we similarly denote with an additional apostrophe[6], as follows for $i \in [M]$.

$$\begin{aligned}
d'_{2,i} &= x'_{i,L/3}, \\
&\quad \text{for } i \in [M - \lceil \log ML \rceil - \lceil \log M \rceil], \\
E_H(\mathbf{d}_2)'_i &= x'_{i+(M-\lceil \log ML \rceil - \lceil \log M \rceil),L/3},
\end{aligned}$$

[6]That is, each one of $\mathbf{d}'_2$, $\mathbf{d}'_4$, etc., is obtained from $\mathbf{d}_2$, $\mathbf{d}_4$, etc., by at most a single substitution.

$$\begin{aligned}
& \text{for } i \in [\lceil \log M \rceil], \\
E_H(\mathbf{s}_1)'_i &= x'_{i+(M-\lceil \log ML \rceil),L/3}, \\
& \text{for } i \in [\lceil \log ML \rceil], \\
d'_{4,i} &= x'_{\tau(i),2L/3}, \\
& \text{for } i \in [M - \lceil \log ML \rceil - \lceil \log M \rceil], \\
E_H(\mathbf{d}_4)'_i &= x'_{\tau(i+(M-\lceil \log ML \rceil - \lceil \log M \rceil)),2L/3}, \\
& \text{for } i \in [\lceil \log M \rceil], \\
E_H(\mathbf{s}_2)'_i &= x'_{\tau(i+(M-\lceil \log ML \rceil)),2L/3}, \\
& \text{for } i \in [\lceil \log ML \rceil], \\
d'_{6,i} &= x'_{\rho(i),L}, \\
& \text{for } i \in [M - \lceil \log ML \rceil - \lceil \log M \rceil], \\
E_H(\mathbf{d}_6)'_i &= x'_{\rho(i+(M-\lceil \log ML \rceil - \lceil \log M \rceil)),L}, \\
& \text{for } i \in [\lceil \log M \rceil], \\
E_H(\mathbf{s}_3)'_i &= x'_{\rho(i+(M-\lceil \log ML \rceil)),L}, \\
& \text{for } i \in [\lceil \log ML \rceil]. \qquad (5.11)
\end{aligned}$$

For example, if we order $\{\mathbf{x}'_i\}_{i=1}^M$ according to τ, then the bottom $\lceil \log(ML) \rceil$ bits of the $(2L/3)$-th column are $E_H(\mathbf{s}_2)'$, the middle $\lceil \log M \rceil$ bits are $E_H(\mathbf{d}_4)'$, and the top $M - \lceil \log ML \rceil - \lceil \log M \rceil$ bits are $\mathbf{d}'_4$ (see Eq. (5.9)). Now, let

$$\begin{aligned}
\mathbf{s}'_1 = (&\hat{\mathbf{a}}_1, \ldots, \hat{\mathbf{a}}_M, \hat{\mathbf{b}}_{\tau(1)}, \ldots, \hat{\mathbf{b}}_{\tau(M)}, \\
&\hat{\mathbf{c}}_{\rho(1)}, \ldots, \hat{\mathbf{c}}_{\rho(M)}), \\
\mathbf{s}'_2 = (&\hat{\mathbf{a}}_{\tau^{-1}(1)}, \ldots, \hat{\mathbf{a}}_{\tau^{-1}(M)}, \hat{\mathbf{b}}_1, \ldots, \hat{\mathbf{b}}_M, \\
&\hat{\mathbf{c}}_{\tau^{-1}\rho(1)}, \ldots, \hat{\mathbf{c}}_{\tau^{-1}\rho(M)}), \quad \text{and} \qquad (5.12) \\
\mathbf{s}'_3 = (&\hat{\mathbf{a}}_{\rho^{-1}(1)}, \ldots, \hat{\mathbf{a}}_{\rho^{-1}(M)}, \hat{\mathbf{b}}_{\rho^{-1}\tau(1)}, \ldots, \hat{\mathbf{b}}_{\rho^{-1}\tau(M)}, \\
&\hat{\mathbf{c}}_1, \ldots, \hat{\mathbf{c}}_M).
\end{aligned}$$

The following lemma shows that at least two of the above $\mathbf{s}'_i$ are close in Hamming distance to their encoded counterpart $(\mathbf{s}_i, E_H(\mathbf{s}_i))$.

Lemma 5.5.1. *There exist distinct integers $k, \ell \in [3]$ such that*

$$\begin{aligned}
&d_H((\mathbf{s}'_k, E_H(\mathbf{s}_k)'), (\mathbf{s}_k, E_H(\mathbf{s}_k)) \le 1, \textit{ and} \\
&d_H((\mathbf{s}'_\ell, E_H(\mathbf{s}_\ell)'), (\mathbf{s}_\ell, E_H(\mathbf{s}_\ell))) \le 1.
\end{aligned}$$

Proof. If the substitution did not occur at either of index sets $\{1,\ldots,L/3-1\}$, $\{L/3+1,\ldots,2L/3-1\}$, or $\{2L/3+1,\ldots,L-1\}$ (which correspond to the values of the strings $\{\mathbf{a}_i\}_{i=1}^M$, $\{\mathbf{b}_i\}_{i=1}^M$, and $\{\mathbf{c}_i\}_{i=1}^M$, respectively), then the orders among the strings $\{\mathbf{a}_i\}_{i=1}^M$, $\{\mathbf{b}_i\}_{i=1}^M$, and $\{\mathbf{c}_i\}_{i=1}^M$ are maintained, respectively. That is, we have that $\tau=\sigma$ and $\rho=\pi$. This implies that

$$\begin{aligned}
\mathbf{s}'_1 = (&\mathbf{a}_1, && \ldots, \mathbf{a}_M, && \mathbf{b}_{\sigma(1)}, && \ldots, \mathbf{b}_{\sigma(M)},\\
&\mathbf{c}_{\pi(1)}, && \ldots, \mathbf{c}_{\pi(M)}),\\
\mathbf{s}'_2 = (&\mathbf{a}_{\sigma^{-1}(1)}, && \ldots, \mathbf{a}_{\sigma^{-1}(M)}, && \mathbf{b}_1, && \ldots, \mathbf{b}_M,\\
&\mathbf{c}_{\sigma^{-1}\pi(1)}, && \ldots, \mathbf{c}_{\sigma^{-1}\pi(M)}),\\
\mathbf{s}'_3 = (&\mathbf{a}_{\pi^{-1}(1)}, && \ldots, \mathbf{a}_{\pi^{-1}(M)}, && \mathbf{b}_{\pi^{-1}\sigma(1)}, && \ldots, \mathbf{b}_{\pi^{-1}\sigma(M)},\\
&\mathbf{c}_1, && \ldots, \mathbf{c}_M),
\end{aligned}$$

and that $d_H(E_H(\mathbf{s}_i),E_H(\mathbf{s}'_i))\le 1$, $i\in[3]$, according to (5.9) and (5.11). In this case, the claim is clear. It remains to show the other cases, and due to symmetry, assume without loss of generality that the substitution occurred in one of the $\mathbf{a}_i$'s, i.e., in an entry which is indexed by an integer in $[L/3-1]$.

Let $A\in\{0,1\}^{M\times L}$ be a matrix whose rows are the $\mathbf{x}_i$'s, in any order. Let A_{left} be the result of ordering the rows of A according to the lexicographic order of their $1,\ldots,L/3-1$ entries. Similarly, let A_{mid} and A_{right} be the results of ordering the rows of A by their $L/3+1,\ldots,2L/3-1$ and $2L/3+1,\ldots,L-1$ entries, respectively, and let A'_{left}, A'_{mid}, and A'_{right} be defined analogously with $\{\mathbf{x}'_i\}_{i=1}^M$ instead of $\{\mathbf{x}_i\}_{i=1}^M$.

It is readily verified that there exist permutation matrices P_1 and P_2 such that $A_{mid}=P_1A_{left}$ and $A_{right}=P_2A_{left}$. Moreover, since $\{\mathbf{b}_i\}_{i=1}^M=\{\hat{\mathbf{b}}_i\}_{i=1}^M$, and $\{\mathbf{c}_i\}_{i=1}^M=\{\hat{\mathbf{c}}_i\}_{i=1}^M$, it follows that $A'_{mid}=P_1(A_{left}+R)$ and $A'_{right}=P_2(A_{left}+R)$, where $R\in\{0,1\}^{M\times L}$ is a matrix of Hamming weight 1; this clearly implies that $A'_{mid}=A_{mid}+P_1R$ and that $A'_{right}=A_{right}+P_2R$. Now, notice that $\mathbf{s}_2$ results from vectorizing some submatrix M_2 of A_{mid}, and $\mathbf{s}'_2$ results from vectorizing some submatrix M'_2 of A'_{mid}. Moreover, the matrices M_2 and M'_2 are taken from their mother matrix by omitting the same rows and columns, and both vectorizing operations consider the entries of M_2 and M'_2 in the same order. In addition, no substitution occurs in the $L/3,\ldots,L$ entries in the $\mathbf{x}_i$'s, which implies that $x'_{\tau i,2L/3}=x_{\pi(i),2L/3}$. Then, the redundancies $E_H(\mathbf{s}_2)'=E_H(\mathbf{s}_2)$ and $E_H(\mathbf{s}_3)'=E_H(\mathbf{s}_3)$ can be identified from (5.11). Therefore, it follows from $A'_{mid}=A_{mid}+P_1R$ that $d_H((\mathbf{s}'_2,E_H(\mathbf{s}'_2)),(\mathbf{s}_2,E_H(\mathbf{s}_2)))\le 1$. The claim for $\mathbf{s}_3$ is similar. □

By applying a Hamming decoder on either one of the $\mathbf{s}_i$'s, the decoder obtains possible candidates for $\{\mathbf{a}_i\}_{i=1}^M$, $\{\mathbf{b}_i\}_{i=1}^M$, and $\{\mathbf{c}_i\}_{i=1}^M$, and by Lemma 5.5.1, it follows that these sets of candidates will coincide in at least two cases. Therefore, the decoder can apply a majority vote of the candidates from the decoding of each $\mathbf{s}_i'$, and the winning values are $\{\mathbf{a}_i\}_{i=1}^M$, $\{\mathbf{b}_i\}_{i=1}^M$, and $\{\mathbf{c}_i\}_{i=1}^M$. Having these correct values, the decoder can sort $\{\mathbf{x}_i'\}_{i=1}^M$ according to their $\mathbf{a}_i$ columns, and deduce the values of σ and π by observing the resulting permutation in the $\mathbf{b}_i$ and $\mathbf{c}_i$ columns, with respect to their lexicographic ordering. This concludes the decoding of the values d_1, d_3, and d_5 of the data d.

We are left to extract d_2, d_4, and d_6. To this end, observe that since the correct values of $\{\mathbf{a}_i\}_{i=1}^M$, $\{\mathbf{b}_i\}_{i=1}^M$, and $\{\mathbf{c}_i\}_{i=1}^M$ are known at this point, the decoder can extract the *true* positions of $\mathbf{d}_2, \mathbf{d}_4$, and $\mathbf{d}_6$, as well as their respective redundancy bits $E_H(\mathbf{d}_2)$, $E_H(\mathbf{d}_4)$, $E_H(\mathbf{d}_6)$. Hence, we have that

$$d_H((\mathbf{d}_i', E_H(\mathbf{d}_i)'), (\mathbf{d}_i, E_H(\mathbf{d}_i))) \leq 1$$

for $i \in \{2,4,6\}$, and thus that the decoding algorithm is complete by applying a Hamming decoder.

We now turn to compute the redundancy of the above code C. Note that there are two sources of redundancy—the Hamming code redundancy, which is at most $3(\log ML + \log M + 2)$ and the fact that the sets $\{\mathbf{a}_i\}_{i=1}^M$, $\{\mathbf{b}_i\}_{i=1}^M$, and $\{\mathbf{c}_i\}_{i=1}^M$ contain distinct strings. By a straightforward computation, for $4 \leq M \leq 2^{L/6}$ we have

$$\begin{aligned}
r(C) =& \log \binom{2^L}{M} \\
&- \log \left(\binom{2^{L/3-1}}{M}^3 \cdot (M!)^2 \cdot 2^{3(M - \log ML - \log M - 2)} \right) \\
=& \log \prod_{i=0}^{M-1} (2^L - i) - \log \prod_{i=0}^{M-1} (2^{L/3-1} - i)^3 \\
&- 3M + 3\log ML + 3\log M + 6 \\
=& \log \prod_{i=0}^{M-1} \frac{(2^L - i)}{(2^{L/3} - 2i)^3} + 3\log ML + 3\log M + 6 \\
\leq& 3M \log \frac{2^{L/3}}{2^{L/3} - 2M} + 3\log ML + 3\log +6. \\
\overset{(a)}{\leq}& 12\log e + 3\log ML + 3\log M + 6,
\end{aligned} \tag{5.13}$$

where inequality (a) is derived in Appendix 5.8.

For the case when $M < \log ML + \log M$, we generate $\{\mathbf{a}_i\}_{i=1}^M$, $\{\mathbf{b}_i\}_{i=1}^M$, and $\{\mathbf{c}_i\}_{i=1}^M$ with length $L/3 - \lceil \frac{\log ML + \log M}{M} \rceil$. As a result, we have $\lceil \frac{\log ML + \log M}{M} \rceil$ bits $x_{i,j}, i \in [M], j \in \{L/3 - \lceil \frac{\log ML + \log M}{M} \rceil + 1, \ldots, L/3\} \cup \{2L/3 - \lceil \frac{\log ML + \log M}{M} \rceil + 1, \ldots, 2L/3\} \cup \{L - \lceil \frac{\log ML + \log M}{M} \rceil + 1, \ldots, L\}$ to accommodate the information bits $\mathbf{d}_2, \mathbf{d}_4, \mathbf{d}_6$ and the redundancy bits $\{E_H(\mathbf{s}_i)\}_{i=1}^3$ and $\{E_H(\mathbf{d}_i)\}_{i \in \{2,4,6\}}$ in each part.

Remark 5.5.1. *The above construction is valid whenever $M \leq 2^{L/3-1}$. However, asymptotically optimal amount of redundancy is achieved for $M \leq 2^{L/6}$.*

Remark 5.5.2. *In this construction, the separate storage of the Hamming code redundancies $E_H(\mathbf{d}_2)$, $E_H(\mathbf{d}_4)$, and $E_H(\mathbf{d}_6)$ is not necessary. Instead, storing $E_H(\mathbf{d}_2, \mathbf{d}_4, \mathbf{d}_6)$ is sufficient, since the true position of those can be inferred after $\{\mathbf{a}_i\}_{i=1}^M$, $\{\mathbf{b}_i\}_{i=1}^M$, and $\{\mathbf{c}_i\}_{i=1}^M$ were successfully decoded. This approach results in redundancy of $3\log ML + \log 3M + O(1)$, and a similar approach can be utilized in the next section as well.*

5.6 Codes for Multiple Substitutions

In this section we extend the 1-substitution correcting code from Sec. 5.5 to multiple substitutions whenever the number of substitutions k is at most $L/(4\log M) - 1/2$. In particular, we obtain the following result.

Theorem 5.6.1. *For integers M, L, and k such that $M \leq 2^{\frac{L}{2(2k+1)}}$ there exists a k-substitution code with redundancy*

$$2k(2k+1)\log ML + 2k(2k+1)\log M + O(k).$$

We restrict our attention to values of M, L, and k for which $2k\lceil \log ML \rceil + 2k\lceil \log M \rceil \leq M$. For the remaining values, i.e., when $2k\lceil \log ML \rceil + 2k\lceil \log M \rceil > M$, a similar code can be constructed. The construction of a k-substitution correcting code is similar in spirit to the single substitution case, except that we partition the strings to $2k+1$ parts instead of 3. In addition, we use a Reed-Solomon code in its binary representation (see Sec. 7.1.1) to combat k-substitutions in the classic sense. The motivation behind considering $2k+1$ parts is that k substitutions can affect at most k of them. As a result, at least $k+1$ parts retain their original order; and that enables a classic RS decoding algorithm to succeed. In turn, the true values of the parts are decided by a majority vote, which is applied over a set of $2k+1$ values, $k+1$ of whom are guaranteed to be correct.

For parameters M, L, and k as above, let

$$D = [\binom{2^{L/(2k+1)-1}}{M}^{2k+1} \cdot (M!)^{2k} \cdot 2^{(2k+1)(M-2klogML-2k\log M)}]$$

be the information set. We split a message $d \in D$ into $d = (d_1, \ldots, d_{4k+2})$, where $d_1 \in [\binom{2^{L/(2k+1)-1}}{M}]$, $d_j \in [\binom{2^{L/(2k+1)-1}}{M}M!]$ for $j \in \{2, \ldots, 2k+1\}$, and $d_j \in [2^{(2k+1)(M-2k\log ML-2k\log M)}]$ for $j \in \{2k+2, \ldots, 4k+2\}$. As in (5.6), we apply F_{com} and F to obtain

$$\begin{aligned} F_{com}(d_1) = \ & \{\mathbf{a}_{1,1}, \ldots, \mathbf{a}_{M,1}\}, \\ & \text{where } \mathbf{a}_{i,1} \in \{0,1\}^{L/(2k+1)-1} \text{ for all } i, \text{ and} \\ F(d_j) = & (\{\mathbf{a}_{1,j}, \ldots, \mathbf{a}_{M,j}\}, \pi_j) \text{ for all } j \in \{2, \ldots, 2k+1\}, \\ & \text{where } \mathbf{a}_{i,j} \in \{0,1\}^{L/(2k+1)-1} \text{ and } \pi_j \in S_M. \end{aligned}$$

As usual, the sets $\{\mathbf{a}_{i,j}\}_{i=1}^M$ are indexed lexicographically according to i, i.e., $\mathbf{a}_{1,j} < \ldots < \mathbf{a}_{M,j}$ for all j. Similar to (5.8), let

$$\begin{aligned} & (x_{i,(j-1)L/(2k+1)+1}, \ldots, x_{i,jL/(2k+1)-1}) \\ = & \mathbf{a}_{\pi_j(i),j}, \ i \in [M], \ j \in [2k+1], \end{aligned}$$

where $\pi_1(i) = i$ is the identity permutation. In addition, define the equivalents of (5.7) as

$$\begin{aligned} \mathbf{s}_1 = (& \mathbf{a}_{1,1}, & \ldots, \ & \mathbf{a}_{M,1}, \\ & \mathbf{a}_{\pi_2(1),2}, & \ldots, \ & \mathbf{a}_{\pi_2(M),2}, \ldots \\ & \mathbf{a}_{\pi_{2k+1}(1),2k+1}, & \ldots, \ & \mathbf{a}_{\pi_{2k+1}(M),2k+1}), \\ \mathbf{s}_2 = (& \mathbf{a}_{\pi_2^{-1}(1),1}, & \ldots, \ & \mathbf{a}_{\pi_2^{-1}(M),1}, \\ & \mathbf{a}_{1,2}, & \ldots, \ & \mathbf{a}_{M,2}, \ldots \\ & \mathbf{a}_{\pi_2^{-1}\pi_{2k+1}(1),2k+1}, & \ldots, \ & \mathbf{a}_{\pi_2^{-1}\pi_{2k+1}(M),2k+1}), \\ & \vdots & & \\ \mathbf{s}_{2k+1} = (& \mathbf{a}_{\pi_{2k+1}^{-1}(1),1}, & \ldots, \ & \mathbf{a}_{\pi_{2k+1}^{-1}(M),1}, \\ & \mathbf{a}_{\pi_{2k+1}^{-1}\pi_2(1),2}, & \ldots, \ & \mathbf{a}_{\pi_{2k+1}^{-1}\pi_2(M),2}, \ldots \\ & \mathbf{a}_{1,2k+1}, & \ldots, \ & \mathbf{a}_{M,2k+1}). \end{aligned}$$

Namely, for every $i \in [2k+1]$, the elements $\{\mathbf{a}_{i,j}\}_{j=1}^M$ appear in $\mathbf{s}_i$ by their lexicographic order, and the remaining ones are sorted accordingly.

To state the equivalent of (5.9), for a binary string $\mathbf{t}$ let $RS_k(\mathbf{t})$ be the redundancy bits that result from k-substitution correcting RS encoding of $\mathbf{t}$, in its binary representation[7]. In particular, we employ an RS code which corrects k substitutions, and incurs $2k\log(|\mathbf{t}|)$ bits of redundancy. Then, the remaining bits $\{x_{i,\frac{L}{2k+1}}\}_{i=1}^{M}$, $\{x_{i,\frac{2L}{2k+1}}\}_{i=1}^{M}, \ldots, \{x_{i,L}\}_{i=1}^{M}$ are defined as follows for $i \in [M]$ and $j \in [1, 2k+1]$.

$$x_{i,\frac{jL}{2k+1}} = \begin{cases} d_{j+2k+1,\pi_j^{-1}(i)} \\ \text{if } \pi_j^{-1}(i) \in [M - 2k\lceil \log M\rceil - 2k\lceil \log ML\rceil] \\ RS_k(\mathbf{d}_{j+2k+1})_{\pi_j^{-1}(i)-M-2k\lceil \log M\rceil - 2k\lceil \log ML\rceil}, \\ \text{if } \pi_j^{-1}(i) \in [M - 2k\lceil \log M\rceil - 2k\lceil \log ML\rceil + 1, \\ \qquad M - 2k\lceil \log ML\rceil] \\ RS_k(\mathbf{s}_j)_{\pi_j^{-1}(i)-M-2k\lceil \log ML\rceil}, \\ \text{if } \pi_j^{-1}(i) \in [M - 2k\lceil \log ML\rceil + 1, M] \end{cases}. \tag{5.14}$$

In this expression, notice that $|\mathbf{s}_i| = M(L - 2k - 1)$ for every i and $|\mathbf{d}_j| \leq M$ for every j. As a result, it follows that $|RS_k(\mathbf{d}_j)| \leq 2k\lceil \log M\rceil$ for every $j \in \{2k+2, \ldots, 4k+2\}$, and $|RS_k(\mathbf{s}_i)| \leq 2k\lceil \log ML\rceil$ for every $i \in [2k+1]$.

To verify that the above construction provides a k-substitution code, observe first that $\{\mathbf{a}_{i,j}\}_{j=1}^{M}$ is a set of distinct integers for all $i \in [2k+1]$, and hence $d_H(\mathbf{x}_i, \mathbf{x}_j) \geq 2k+1$ for all distinct i and j in $[M]$. Thus, a k-substitution error cannot turn one $\mathbf{x}_i$ into another, and the result is always in $\binom{\{0,1\}^L}{M}$.

The decoding procedure also resembles the one in Sec. 5.5. Upon receiving a word $C' = \{\mathbf{x}'_1, \ldots, \mathbf{x}'_M\} \in \mathcal{B}_k(C)$ for some codeword C, where the indexing of the elements of C' is lexicographic, we define

$$\hat{\mathbf{a}}_{i,j} = (x'_{\tau_j^{-1}(i),\frac{(j-1)L}{2k+1}+1}, \ldots, x'_{\tau_j^{-1}(i),\frac{jL}{2k+1}-1}),$$
$$\text{for } j \in [2k+1], \text{ and } i \in [M]$$

where τ_j is the permutation by which $\{\mathbf{x}'_i\}_{i=1}^{M}$ are sorted according to their $\frac{(j-1)L}{2k+1} + 1, \ldots, \frac{jL}{2k+1} - 1$ entries (τ_1 is the identity permutation, compared with (5.10)). In

[7]To avoid uninteresting technical details, it is assumed henceforth that RS encoding in its binary form is possible, i.e., that $\log(|\mathbf{t}|)$ is an integer that divides $\mathbf{t}$; this can always be attained by padding with zeros. Furthermore, the existence of an RS code is guaranteed, since $q = 2^{\log(|\mathbf{t}|)}$ is larger than the length of the code, which is $|\mathbf{t}|/\log(|\mathbf{t}|)$.

addition, sorting $\{\mathbf{x}'_i\}_{i=1}^{M}$ by either one of τ_j yields *candidates* for $\{RS_k(\mathbf{s}_i)\}_{i=1}^{2k+1}$, for $\{\mathbf{d}_j\}_{j=2k+2}^{4k+2}$, and for $\{RS_k(\mathbf{d}_j)\}_{j=2k+2}^{4k+2}$, as follows.

$$\begin{aligned} d'_{j+2k+1,i} &= x'_{\pi_j(i),\frac{jL}{2k+1}}, \\ &\quad \text{for } i \in [M - 2k\lceil\log ML\rceil - 2k\lceil\log M\rceil] \\ RS_k(\mathbf{d}_{j+2k+1})'_i &= x'_{\pi_j(i+(M-2k\lceil\log ML\rceil-2k\lceil\log M\rceil)),\frac{jL}{2k+1}}, \\ &\quad \text{for } i \in [2k\lceil\log M\rceil] \\ RS_k(\mathbf{s}_j)'_i &= x'_{\pi_j(i+(M-2k\lceil\log ML\rceil)),\frac{jL}{2k+1}}, \\ &\quad \text{for } i \in [2k\lceil\log ML\rceil] \end{aligned} \tag{5.15}$$

The respective $\{\mathbf{s}'_i\}_{i=1}^{2k+1}$ are defined as

$$\begin{aligned} \mathbf{s}'_1 = (&\hat{\mathbf{a}}_{1,1}, &&\ldots, \hat{\mathbf{a}}_{M,1}, \\ &\hat{\mathbf{a}}_{\tau_2(1),2}, &&\ldots, \hat{\mathbf{a}}_{\tau_2(M),2}, \ \ldots \\ &\hat{\mathbf{a}}_{2k+1,\tau_{2k+1}(1)}, &&\ldots, \hat{\mathbf{a}}_{2k+1,\tau_{2k+1}(M)}, \\ &RS_k(\mathbf{s}_1)', \\ &\hat{\mathbf{a}}_{\tau_{2k+1}(1),2k+1}, &&\ldots, \hat{\mathbf{a}}_{\tau_{2k+1}(M),2k+1}), \\ \mathbf{s}'_2 = (&\hat{\mathbf{a}}_{\tau_2^{-1}(1),1}, &&\ldots, \hat{\mathbf{a}}_{\tau_2^{-1}(M),1}, \\ &\hat{\mathbf{a}}_{1,2}, &&\ldots, \hat{\mathbf{a}}_{M,2}, \ \ldots \\ &\hat{\mathbf{a}}_{2k+1,\tau_2^{-1}\tau_{2k+1}(1)}, &&\ldots, \hat{\mathbf{a}}_{2k+1,\tau_2^{-1}\tau_{2k+1}(M)}, \\ &RS_k(\mathbf{s}_2)', \\ &\hat{\mathbf{a}}_{\tau_2^{-1}\tau_{2k+1}(1),2k+1}, &&\ldots, \hat{\mathbf{a}}_{\tau_2^{-1}\tau_{2k+1}(M),2k+1}), \\ &\vdots \\ \mathbf{s}'_{2k+1} = (&\hat{\mathbf{a}}_{\tau_{2k+1}^{-1}(1),1}, &&\ldots, \hat{\mathbf{a}}_{\tau_{2k+1}^{-1}(M),1}, \\ &\hat{\mathbf{a}}_{\tau_{2k+1}^{-1}\tau_2(1),2}, &&\ldots, \hat{\mathbf{a}}_{\tau_{2k+1}^{-1}\tau_2(M),2}, \ \ldots \\ &\hat{\mathbf{a}}_{2k+1,1}, &&\ldots, \hat{\mathbf{a}}_{2k+1,M}, \\ &RS_k(\mathbf{s}_{2k+1})', \\ &\hat{\mathbf{a}}_{1,2k+1}, &&\ldots, \hat{\mathbf{a}}_{M,2k+1}). \end{aligned}$$

Lemma 5.6.1. *There exist $k+1$ distinct integers $\ell_1, \ldots, \ell_{k+1}$ such that*

$$d_H((\mathbf{s}'_{\ell_j}, RS_k(\mathbf{s}_{\ell_j})'), (\mathbf{s}_{\ell_j}, RS_k(\mathbf{s}_{\ell_j}))) \le k$$

for every $j \in [k+1]$.

Proof. Analogous to the proof of Lemma 5.5.1. See Appendix 5.8 for additional details. □

By applying an RS decoding algorithm on each of $\{\mathbf{s}'_i\}_{i=1}^{2k+1}$ we obtain candidates for the true values of $\{\mathbf{a}_{i,j}\}_{j=1}^{M}$ for every $i \in [2k+1]$. According to Lemma 5.6.1, at least $k+1$ of these candidate coincide, and hence the true value of $\{\mathbf{a}_{i,j}\}_{j=1}^{M}$ can be deduced by a majority vote. Once these true values are known, the decoder can sort $\{\mathbf{x}'_i\}_{i=1}^{M}$ by its $\mathbf{a}_{1,j}$ entries (i.e., the entries indexed by $1, \ldots, \frac{L}{2k+1} - 1$), and deduce the values of each π_t, $t \in \{2, \ldots, 2k+1\}$ according to the resulting permutation of $\{\mathbf{a}_{t,\ell}\}_{\ell=1}^{M}$ in comparison to their lexicographic one. Having all the permutations $\{\pi_j\}_{j=2}^{2k+1}$, the decoder can extract the true positions of $\{\mathbf{d}_j\}_{j=2k+2}^{4k+2}$ and $\{RS_k(\mathbf{d}_j)\}_{j=2k+2}^{4k+2}$, and apply an RS decoder to correct any substitutions that might have occurred.

Remark 5.6.1. *Notice that the above RS code in its binary representation consists of binary substrings that represent elements in a larger field. As a result, this code is capable of correcting* any *set of substitutions that are confined to at most k of these substrings. Therefore, our code can correct more than k substitutions in many cases.*

For $4 \le M \le 2^{L/2(2k+1)}$, the total redundancy of the above construction C is given by

$$\begin{aligned} r(C) =& \log \binom{2^L}{M} - \log \binom{2^{L/(2k+1)-1}}{M}^{2k+1} \\ &- \log(M!^{2k} 2^{(2k+1)(M-2k\log ML-2k\log M)}) \\ &\overset{(b)}{\le} (2k+1)\log e + 2k(2k+1)\log ML \\ &+ 2k(2k+1)\log M, \end{aligned} \tag{5.16}$$

where the proof of inequality (b) is given in Appendix 5.8.

Remark 5.6.2. *As mentioned in Remark 5.5.2, storing $RS_k(\mathbf{d}_j)$ separately in each part $j \in \{2k+2, \ldots, 4k+2\}$ is not necessary. Instead, we store $RS_k(\mathbf{d}_{2k+2}, \ldots, \mathbf{d}_{4k+2})$ in a single part $j = 2k+1$, since the position of the binary strings $\mathbf{d}_j$ for $j \in \{2k+2, \ldots, 4k+2\}$ and the redundancy $RS_k(\mathbf{d}_{2k+2}, \ldots, \mathbf{d}_{4k+2})$ can be identified once $\{\mathbf{a}_{i,j}\}_{i \le M, j \le 2k+1}$ are determined. The redundancy of the resulting code is $2k(2k+1)\log ML + 2k\log(2k+1)M$.*

For the case when $M < 2k\log ML + 2k\log M$, we generate sequences $\mathbf{a}_{i,j}$, $i \in [M]$, $j \in [2k+1]$ with length $L/(2k+1) - \lceil \frac{2k\log ML+2k\log M}{M} \rceil$. Then, the length $\lceil \frac{2k\log ML+2k\log M}{M} \rceil$ sequences $x_{i,j}$, $i \in [M]$, $j \in \cup_{l=1}^{2k+1}\{(l-1)L/(2k+1) - \lceil \frac{2k\log ML+2k\log M}{M} \rceil + 1, \ldots, lL/(2k+1)\}$ are used to accommodate the information bits $\{\mathbf{d}_j\}_{j=2k+2}^{4k+2}$ and the redundancy bits $\{RS_k(\mathbf{s}_i)\}_{i=1}^{2k+1}$ and $\{RS_k(\mathbf{d}_j)\}_{j=2k+2}^{4k+2}$ in each part.

5.7 Conclusions and Future Work

Motivated by novel applications in coding for DNA storage, this chapter presented a channel model in which the data is sent as a set of unordered strings that are distorted by substitutions. Respective sphere packing arguments were applied in order to establish an existence result of codes with low redundancy for this channel, and a corresponding lower bound on the redundancy for $k = 1$ was given by using Fourier analysis. For $k = 1$, a code construction was given which asymptotically achieves the lower bound. For larger values of k, a code construction whose redundancy is asymptotically k times the aforementioned upper bound was given; closing this gap is an interesting open problem. Furthermore, it is intriguing to find a lower bound on the redundancy for larger values of k as well.

5.8 Appendix

Proof of Lemma 5.6.1

Proof. (of Lemma 5.6.1) Similarly to the proof of Lemma 5.5.1, we consider a matrix $A \in \{0,1\}^{M\times L}$ whose rows are the $\mathbf{x}_i$'s, in any order. Let A_j be the result of ordering the rows of A according to the lexicographic order of their $(j-1)L/(2k+1)+1, \ldots, jL/(2k+1)-1$ bits for $j \in [2k+1]$. The matrices A'_j for $j \in [2k+1]$ can be defined analogously with $\{\mathbf{x}'_i\}_{i=1}^M$ instead of $\{\mathbf{x}_i\}_{i=1}^M$.

It is readily verified that there exist $2k+1$ permutation matrices P_j such that $A_j = P_j A$ (Here P_1 is the identity matrix). Moreover, since k substitution spoils at most k parts, there exist at least $j_l \in [2k+1], l \in [k+1]$ such that $\{\mathbf{a}_{i,j_l}\}_{i=1}^M = \{\mathbf{a}_{\hat{i,j_l}}\}_{i=1}^M$, for $l \in [k+1]$, it follows that $A'_{j_l} = P_{j_l}(A+R)$ for $l \in [k+1]$, where $R \in \{0,1\}^{M\times L}$ is a matrix of Hamming weight at most k; this clearly implies that $A'_{j_l} = A_{j_l} + P_{j_l}R$ for $l \in [k+1]$. Since $\mathbf{s}_{j_l}$ results from vectorizing some submatrix M_l of A_{j_l}, and $\mathbf{s}'_{j_l}$ results from vectorizing some submatrix M'_l of A'_{j_l}. Moreover, the matrices M_l and M'_l are taken from their mother matrix by omitting the same rows and columns, and both vectorizing operations consider the entries of M_l and M'_l in the same order. In addition, the redundancies $E_H(\mathbf{s}_{j_l})$ for $l \in [k+1]$ can be identified similarly, and have at most k substitution with respect to the corresponding entries in the noiseless codeword. Therefore, it follows from $A_{j_l} = A_{j_l} + P_1R$ that $d_H((\mathbf{s}'_{j_l}, E_H(\mathbf{s}_{j_l})'), (\mathbf{s}_{j_l}, E_H(\mathbf{s}_{j_l}))) \leq k$.

□

Proof of Redundancy Bounds

Proof of (a) *in* (5.13)*:*

$$\begin{aligned}
r(C) &\leq 3\log(1+\frac{2M}{2^{L/3}-2M})^M + 3\log ML + 3\log M + 6\\
&\leq 3\log(1+\frac{4}{M})^M + 3\log ML + 3\log M + 6\\
&= 12\log((1+\frac{4}{M})^{M/4}) + 3\log ML + 3\log M + 6\\
&\leq 12\log e + 3\log ML + 3\log M + 6.
\end{aligned}$$

Proof of (b) *in* (5.16)*:*

$$\begin{aligned}
r(C) =& \log\prod_{i=0}^{M-1}(2^L - i) - \log\prod_{i=0}^{M-1}(2^{L/(2k+1)-1} - i)^{2k+1}\\
&- \log 2^{(2k+1)M} + 2k(2k+1)\log ML\\
&+ 2k(2k+1)\log M\\
=& \log\prod_{i=0}^{M-1}\frac{(2^L - i)}{(2^{L/(2k+1)} - 2i)^{2k+1}}\\
&+ 2k(2k+1)\log ML + 2k(2k+1)\log M\\
\leq& (2k+1)M\log\frac{2^{L/(2k+1)}}{2^{L/(2k+1)} - 2M}\\
&+ 2k(2k+1)\log ML + 2k(2k+1)\log M\\
\leq& (2k+1)\log(1+\frac{2M}{2^{L/(2k+1)} - 2M})^M\\
&+ 2k(2k+1)\log ML + 2k(2k+1)\log M\\
\leq& (2k+1)\log(1+\frac{4}{M})^M + 2k(2k+1)\log ML\\
&+ 2k(2k+1)\log M\\
=& (2k+1)\log((1+\frac{4}{M})^{M/4}) + 2k(2k+1)\log ML\\
&+ 2k(2k+1)\log M\\
\leq& (2k+1)\log e + 2k(2k+1)\log ML
\end{aligned}$$

$$+ 2k(2k + 1)\log M.$$

Improved Codes for a Single Substitution

We briefly present an improved construction of a single substitution code, which achives $2\log ML + \log 2M + O(1)$ redundancy.

Theorem 5.8.1. *Let M and L be numbers that satisfy $M \le 2^{L/4}$. Then there exists a single substitution correcting code with redundancy $2\log ML + \log 2M + O(1)$.*

The construction is based on the single substitution code as shown in Sec. 5.5. The difference is that instead of using three parts and the majority rule, it suffices to use two parts (two halfs) and an extra bit to indicate which part has the correct order. To compute this bit, let

$$\mathbf{x}_{\oplus} = \bigoplus_{i=1}^{M} \mathbf{x}_i$$

be the bitwise XOR of all strings $\mathbf{x}_i$ and $\mathbf{e} \in \{0,1\}^L$ be a vector of $L/2$ zeros followed by $L/2$ ones. We use the bit $b_e = \mathbf{e} \cdot \mathbf{x}_{\oplus} \bmod 2$ to indicate in which part the substitution error occurs. If a substitution error happens at the first half $(x_i^1, \ldots, x_i^{L/2})$, the bit b_e does not change. Otherwise the bit b_e is flipped. Moreover, as mentioned in Remark 5.5.2, we store the redundancy of all the binary strings in a single part, instead of storing the redundancy separately for each binary string in each part. The data to encode is regarded as $d = (d_1, d_2, d_3, d_4)$, where $d_1 \in [\binom{2^{L/2-1}}{M}]$, $d_2 \in [\binom{2^{L/2-1}}{M} \cdot M!]$, $d_3 \in [2^{M-\log ML-1}]$ and $d_4 \in [2^{M-\log ML-\log 2M-2}]$. That is, d_1 represents a set of M strings of length $L/2-1$, d_2 represents a set of M strings of length $L/2-1$ and a permutation π. Let $\mathbf{d}_3 \in \{0,1\}^{M-\log ML-1}, \mathbf{d}_4 \in \{0,1\}^{M-\log ML-\log 2M-2}$ be the binary strings corresponding to d_3 and d_4 respectively.

We now address the problem of inserting the bit b_e into the codeword. We consider the four bits $x_{i_1,L/2}, x_{i_2,L/2}, x_{i_3,L}$, and $x_{i_4,L}$, where i_1 and i_2 are the indices of the two largest strings among $\{\mathbf{a}_i\}_{i=1}^M$ in lexicographic order, and i_3 and i_4 are the indices of the two largest strings among $\{\mathbf{b}_i\}_{i=1}^M$ in lexicographic order. Then, we compute b_e and set

$$x_{i_1,L/2} = x_{i_2,L/2} = x_{i_3,L} = x_{i_4,L} = b_e.$$

Note that after a single substitution, at most one of i_1, i_2, i_3, and i_4 will not be among the indices of the largest two strings in their corresponding part. Hence, upon receiving a word $C' = \{\mathbf{x}'_1, \ldots, \mathbf{x}'_M\} \in \mathcal{B}_1(C)$ for some codeword C, we find the two largest strings among $\{\mathbf{a}_i\}_{i=1}^M$ and the two largest strings among $\{\mathbf{b}_i\}_{i=1}^M$, and use majority to determine the bit b_e. The rest of the encoding and decoding procedures are similar to the corresponding ones in Sec. 5.5. We define $\mathbf{s}_1$ and $\mathbf{s}_2$ to the two possible concatenations of $\{\mathbf{a}_i\}_{i=1}^M$ and $\{\mathbf{b}_i\}_{i=1}^M$,

$$\begin{aligned}
\mathbf{s}_1 &= (\mathbf{a}_1, \quad \ldots, \mathbf{a}_M, \quad \mathbf{b}_{\pi(1)}, \ldots, \mathbf{b}_{\pi(M)}) \\
\mathbf{s}_2 &= (\mathbf{a}_{\pi^{-1}(1)}, \ldots, \mathbf{a}_{\pi^{-1}(M)}, \mathbf{b}_1, \quad \ldots, \mathbf{b}_M).
\end{aligned}$$

We compute their Hamming redundancies and place them in columns $L/2$ and L, alongside the strings d_3, d_4 and their Hamming redundancy $E_H(\mathbf{d}_3, \mathbf{d}_4)$ in column L, similar to (5.9).

In order to decode, we compute the value of b_e by a majority vote, which locates the substitution, and consequently, we find π by ordering $\{\mathbf{x}'_i\}_{i=1}^M$ according to the error-free part. Knowing π, we extract the d_i's and their redundancy $E_H(\mathbf{d}_3, \mathbf{d}_4)$, and complete the decoding procedure by applying a Hamming decoder. The resulting redundancy is $2 \log ML + \log 2M + 3$.

5.9 Proof of (a) in Eq. (5.5)

Note that $P \leq T$, it suffices to show that the function $g(P) \triangleq ((T + P)/P)^P = (1 + T/P)^P$ is increasing in P for $P > 0$. We now show that the derivative $\partial g(P)/\partial P = (1 + T/P)^P (\ln(1 + T/P) - T/(T + P))$ is greater than 0 for $P > 0$. It is left to show that

$$\ln(1 + T/P) > T/(T + P). \tag{5.17}$$

Let $v = T/(T + P)$, then Eq. (5.17) is equivalent to

$$1/(1 - v) > e^v \tag{5.18}$$

for some $0 < v < 1$. The inequality (5.18) holds since $1/(1 - v) = 1 + \sum_{i=1}^{\infty} v^i$ and $e^v = 1 + \sum_{i=1}^{\infty} v^i/i!$ for $0 < v < 1$.

Chapter 6

ROBUST INDEXING: OPTIMAL CODES CORRECTING DELETION/INSERTION AND SUBSTITION ERRORS

In this chapter, we consider a more general setting of correcting deletions/insertions over unordered sets. Our construction in this chapter improved the one in Ch. 5 and is order-wise optimal.

6.1 Introduction

This chapter is a follow-up work of Ch. 6. Consider encoding data into M strings of length L. The decoder wishes to recover the data from erroneous versions of the M strings, which contain substitution, deletion, and insertion errors. This model has been extensively investigated. In Ch. 5, we showed that for a constant number k of substitution errors, the optimal redundancy has the order $O(k \log ML)$ and an explicit code with $O(k^2 \log ML)$ bits of redundancy was given. The problem of designing codes correcting a constant number of substitutions was also studied in [89], from a generalized Hamming distance perspective. Yet no order optimal code construction for substitution errors was given.

In this chapter, we propose order-wise optimal code constructions that achieves $O(k \log ML)$ redundancy for k substitution errors, based on a technique called *robust indexing*. Our first main result is as follows.

Theorem 6.1.1. *For integers M, L, and k, let $L' \triangleq 3 \log M + 4k^2 + 1$. If $L' + 4kL' + 2k \log(4kL') \leq L$, then there exists an explicit k-substitution code, computable in $poly(M, L, k)$ time, that has redundancy $2k \log ML + (12k + 2) \log M + O(k^3) + O(k \log \log ML)$.*

Instead of assigning index directly as in index based coding, we *embed information into the index*. Note that to combat errors, the index bits themselves form a substitution code and information is carried through choices of the code. Our *robust indexing* algorithm generates indexing bits in a greedy manner and has polynomial complexity.

Our algorithm also applies to a combination of deletion and insertion errors with slight modification. Note that correcting a combination of deletion/insertion errors is harder than correcting substitution errors only, since a substitution error is a deletion error followed by an insertion error. Very few works studied the problem of correcting a combination of deletion/insertion errors, the most related one being [62], which considered correcting a single deletion. With our deletion/insertion correcting codes in Ch. 4, we are able to present a code that corrects a combination of k deletions and insertions with $O(k \log ML)$ redundancy, which is our second main result.

Theorem 6.1.2. *For integers M, L, k, and $L' \triangleq 3\log M + 4k^2 + 1$. If $L' + 4kL' + 2k\log(4kL') \leq L$, then there exists a code, computable in $poly(M, L)$ time, that corrects a combination of at most k deletions and insertions in total, with redundancy $4k\log ML + (12k+2)\log M + O(k^3) + o(\log ML)$.*

The rest of this chapter is organized as follows. Sec. 6.2 presents the notations and channel model. In Sec. 6.3 we provide an order optimal code construction for substitution errors, and the *robust indexing* algorithm is given in Sec. 6.4. In Sec. 6.5 we apply *robust indexing* to deletion errors and propose a deletion/insertion correcting code construction.

6.2 Preliminaries

We focus on the binary alphabet $\{0, 1\}$. With a slight abuse of notation, for a set S and an integer m, denote by $\binom{S}{m}$ the family of all sets of m different elements in S, and by $\binom{S}{\leq m} = \bigcup_{i=1}^{M} \binom{S}{m}$ the family of all subsets of S with no more than m elements. For an integer ℓ, let $\{0, 1\}^{\leq \ell}$ be the set of all binary strings with length at most ℓ. In our channel model, it is assumed that the data is given as a binary string and encoded in an unordered set of M different strings $\{\mathbf{x}_i\}_{i=1}^{M}$ of length L. Hence in this chapter, a codeword refers to a set $\{\mathbf{x}_i\}_{i=1}^{M} \in \binom{\{0,1\}^L}{M}$, rather than a vector as in classic coding theoretic settings. Each element $\mathbf{x}_i$ in a codeword is referred to as a string. The assumption that the strings $\mathbf{x}_i$, $i \in [M]$, in a codeword are different stems from the fact that sequencing procedures cannot detect repeated strings in the codeword $\{\mathbf{x}_i\}_{i=1}^{M}$ by counting the frequency of each string in the sample. Moreover, as we can see from the definition of code redundancy that will be presented later, the asymptotic redundancy of a code is not affected by allowing repeated string in the codeword, when $M = o(2^L)$.

The codeword $\{\mathbf{x}_i\}_{i=1}^{M}$ is subject to substitution, deletion, and insertion errors. In this chapter, we propose codes for correcting substitution errors and deletion errors separately. The presented deletion codes are capable of correcting deletion/insertion errors as well. A k-substitution error is an operation that flips at most k bits in the codeword. Each bit flip can occur in any of the strings $\mathbf{x}_i, i \in [M]$, where $[M] \triangleq \{1, \ldots, M\}$. For any string set $\{\mathbf{x}_i\}_{i=1}^M \in \binom{\{0,1\}^L}{M}$, define its Hamming ball $\mathcal{B}_k^H(\{\mathbf{x}_i\}_{i=1}^M) \in \binom{\{0,1\}^L}{M}$ as the set of all possible outcomes after a k substitution error in $\{\mathbf{x}_i\}_{i=1}^M$. A k substitution code C^H is an ensemble of codewords $\{\mathbf{x}_i\}_{i=1}^M \in \binom{\{0,1\}}{M}$ such that for any $S_1, S_2 \in C^H$, we have that $\mathcal{B}_k^H(S_1) \cap \mathcal{B}_k^H(S_2) = \emptyset$. Similarly, a k deletion/insertion error is an operation that removes and/or inserts at most k bits in the codeword. For a set $\{\mathbf{x}_i\}_{i=1}^M \in \binom{\{0,1\}^L}{M}$, its deletion ball $\mathcal{B}_k^D(\{\mathbf{x}_i\}_{i=1}^M) \subseteq \binom{\{0,1\}^{\leq L}}{\leq M}$ is the collection of outputs resulted from a k deletion/insertion error in $\{\mathbf{x}_i\}_{i=1}^M$. A k deletion/insertion code C^D consists of sets $\{\mathbf{x}_i\}_{i=1}^M \in \binom{\{0,1\}^L}{M}$ such that the deletion/insertion balls of any $S_1, S_2 \in C^D$ do not intersect. The redundancy of a k substitution or deletion/insertion code C is defined as $r(C) = \binom{2^L}{M} - \log |C|$.

Our code constructions make use of the well-known Reed-Solomon code, which is capable of correcting k substitutions in a length n codeword over an alphabet of size q, with $2k \log q$ bits redundancy, as long as $q \geq n - 1$ [79]. Moreover, combinatorial numbering maps [53] are used in the robust indexing algorithm. Specifically, for integers m and n, there exist a map $F_{com} : [\binom{n}{m}] \to \binom{[n]}{m}$ that maps an integer $d \in [\binom{n}{m}]$ to a set of m different elements in $[n]$, and a map $F_{perm} : [n!] \to S_n$ that maps an integer $d \in [n!]$ into a permutation on n elements.

6.3 Robust Indexing for Codes over Sets: Substitution Errors

In this section we describe our code constructions, which use the idea of robust indexing. These codes deal with substitution errors in the M strings. Our codes have redundancy $O(k \log ML)$, which is order-wise optimal whenever k is at most $O(\min\{L^{1/3}, L/\log M\})$.

Theorem 6.3.1. *For integers M, L, and k, let $L' \triangleq 3 \log M + 4k^2 + 1$. If $L' + 4kL' + 2k \log(4kL') \leq L$, then there exists an explicit k-substitution code, computable in $poly(M, L, k)$ time, that has redundancy $2k \log ML + (12k + 2) \log M + O(k^3) + O(k \log \log ML)$.*

Since the codewords consist of unordered strings, we assign indexing bits to each string such that the strings are ordered. However, instead of directly assigning the indices $1, \ldots, M$ to each string, we embed information into the indexing bits. In other words, we use the information bits themselves for the purpose of indexing. This provides more efficiency in sending information.

Specifically, for a codeword $W = \{\mathbf{x}_i\}_{i=1}^M$, we choose the first L' bits $(x_{i,1}, x_{i,2}, \ldots, x_{i,L'})$, $i \in [M]$ in each string $\mathbf{x}_i$ as the indexing bits, and encode information in them. Then, the strings $\{\mathbf{x}_i\}_{i=1}^M$ are sorted according to the lexicographic order π of the indexing bits $(x_{i,1}, x_{i,2}, \ldots, x_{i,L'})$, $i \in [M]$, where $(x_{\pi(i),1}, x_{\pi(i),2}, \ldots, x_{\pi(i),L'}) < (x_{\pi(j),1}, x_{\pi(j),2}, \ldots, x_{\pi(j),L'})$ for $i < j$. Once $\{\mathbf{x}_i\}_{i=1}^M$ are ordered, it suffices to use a Reed-Solomon code to protect the concatenated string $(\mathbf{x}_{\pi(1)}, \ldots, \mathbf{x}_{\pi(M)})$, and thus the codeword $\{\mathbf{x}_i\}_{i=1}^M$, from k substitution errors.

One of the key issues with this approach is that the indexing bits and their lexicographic order can be disrupted by substitution errors. To deal with this, we present a technique referred to as *robust indexing*, which protects the indexing bits from substitution errors. The basic ideas of *robust indexing* are as follows: (1) Constructing the indexing bits $\{(x_{i,1}, x_{i,2}, \ldots, x_{i,L'})\}_{i=1}^M$ such that the Hamming distance between any two distinct $(x_{i,1}, x_{i,2}, \ldots, x_{i,L'})$ and $(x_{j,1}, x_{j,2}, \ldots, x_{j,L'})$ is at least $2k+1$, i.e., the strings $\{(x_{i,1}, x_{i,2}, \ldots, x_{i,L'})\}_{i=1}^M$ form a error correcting code under classic coding theoretic definition. Then, we can identify which string among $\{(x_{i,1}, x_{i,2}, \ldots, x_{i,L'})\}_{i=1}^M$ results in the erroneous version $(x'_{i,1}, x'_{i,2}, \ldots, x'_{i,L'})$, by using a minimum Hamming distance criterion; (2) Using additional redundancy to protect the set of indexing bits $\{(x_{i,1}, x_{i,2}, \ldots, x_{i,L'})\}_{i=1}^M$ from substitution errors. Note that we encode data in the code $\{(x_{i,1}, x_{i,2}, \ldots, x_{i,L'})\}_{i=1}^M$ through different choices of the code. After substitution errors, two choices of the code, which represent different messages, might result in the same read $\{(x_{i,1}, x_{i,2}, \ldots, x_{i,L'})\}_{i=1}^M$.

Example 6.3.1. *For $k = M = 2$ and $L = 8$, consider two codes $\{11111111, 00000000\}$ and $\{11111111, 00010010\}$. Both have minimum Hamming distance greater than $2k+1 = 5$ and can result in the same set $\{11111111, 00000011\}$ after $k = 2$ substitutions.*

Hence, to recover the indexing bits $(x_{i,1}, x_{i,2}, \ldots, x_{i,L'})$, $i \in [M]$, we need to know the code $\{(x_{i,1}, x_{i,2}, \ldots, x_{i,L'})\}_{i=1}^M$ to which the erroneous string $(x'_{i,1}, x'_{i,2}, \ldots, x'_{i,L'})$ is corrected, $i \in [M]$.

For an integer ℓ, let $\mathbb{1}_\ell$ be the all 1's vector of length ℓ. Define $\mathcal{S}^H$ as the set of all length L' codes with cardinality M and minimum Hamming distance at least $2k+1$, which contain $\mathbb{1}_{L'}$, that is,

$$\mathcal{S}^H \triangleq \Big\{\{\mathbf{a}_1, \ldots, \mathbf{a}_M\} \in \binom{\{0,1\}^{L'}}{M} \Big| \mathbf{a}_1 = \mathbb{1}_{L'} \text{ and } d_H(\mathbf{a}_i, \mathbf{a}_j) \geq 2k+1$$
$$\text{for every distinct } i, j \in [M]\Big\}.$$

The following lemma gives a lower bound on the size of $\mathcal{S}^H$ and is obtained using counting argument.

Lemma 6.3.1. *Let $Q = \sum_{i=0}^{2k} \binom{L'}{i}$ be the size of a Hamming ball of radius $2k$ centered at a vector in $\{0,1\}^{L'}$. We have that*

$$|\mathcal{S}^H| \geq \frac{(2^{L'} - MQ)^{M-1}}{(M-1)!}. \tag{6.1}$$

Proof. Define the set of ordered tuples

$$\mathcal{S}_T^H = \left\{(\mathbf{a}_1, \ldots, \mathbf{a}_M) : \mathbf{a}_1 = \mathbb{1}_{L'} \text{ and } d_H(\mathbf{a}_i, \mathbf{a}_j) \geq 2k+1 \text{ for distinct } i, j \in [M]\right\}$$

such that for each tuple $(\mathbf{a}_1, \ldots, \mathbf{a}_M) \in \mathcal{S}_T^H$, we have that $\{\mathbf{a}_1, \ldots, \mathbf{a}_M\} \in \mathcal{S}^H$. We show that $|\mathcal{S}_T^H| \geq \prod_{i=2}^{M}[2^{L'} - (i-1)Q]$, by finding $\prod_{i=2}^{M}[2^{L'} - (i-1)Q]$ tuples in $\mathcal{S}_T^H$. Let $\mathbf{a}_1 = \mathbb{1}_{L'}$. We select $\mathbf{a}_2, \ldots, \mathbf{a}_M$ sequentially such that each selected string $\mathbf{a}_i$, $i \in [2, M] \triangleq \{2, \ldots, M\}$, is of Hamming distance at least $2k+1$ from each one of $\mathbf{a}_1, \ldots, \mathbf{a}_{i-1}$. The tuple $(\mathbf{a}_1, \ldots, \mathbf{a}_M)$ selected in this way has pairwise Hamming distance at least $2k+1$ and thus belongs to $\mathcal{S}_T^H$.

Since the number of strings having Hamming distance at most $2k$ from at least one of $\mathbf{a}_1, \ldots, \mathbf{a}_{i-1}$ is at most $(i-1)Q$, there are at least $2^{L'} - (i-1)Q$ possible choices of $\mathbf{a}_i$ that have Hamming distance at least $2k+1$ from each one of $\mathbf{a}_1, \ldots, \mathbf{a}_{i-1}$. Therefore, the total number of ways selecting tuples $(\mathbf{a}_1, \ldots, \mathbf{a}_M)$ is at least $\prod_{i=2}^{M}[2^{L'} - (i-1)Q]$. Since in the above selection of tuples $(\mathbf{a}_1 = \mathbb{1}_{L'}, \ldots, \mathbf{a}_M) \in \mathcal{S}_T^H$, there are $(M-1)!$ tuples that correspond to the same set $\{\mathbf{a}_1 = \mathbb{1}_{L'}, \mathbf{a}_2, \ldots, \mathbf{a}_M\}$ in $\mathcal{S}^H$, we have that

$$|\mathcal{S}^H| = |\mathcal{S}_T^H|/(M-1)! \geq \prod_{i=2}^{M}[2^{L'} - (i-1)Q]/(M-1)! \geq \frac{(2^{L'} - MQ)^{M-1}}{(M-1)!}.$$

□

According to (6.1), there exists an invertible mapping $F_S^H : [\lceil \frac{(2^{L'}-MQ)^{M-1}}{(M-1)!} \rceil] \rightarrow \binom{\{0,1\}^{L'}}{M}$, computed in $O(2^{ML'})$ time using brute force, that maps an integer $d \in \left[\lceil \frac{(2^{L'}-MQ)^{M-1}}{(M-1)!} \rceil\right]$ to a code $F_S^H(d) \in \mathcal{S}^H$. In the next session, we will present a polynomial time algorithm that computes a map $F_S^H(d)$ for any $d \in \left[\lceil \frac{(2^{L'}-MQ)^{M-1}}{(M-1)!} \rceil\right]$ such that $F_S^H(d) \in \mathcal{S}^H$ and $F_S^H(d_1) \neq F_S^H(d_2)$ for $d_1 \neq d_2$ and $d_1, d_2 \in \left[\lceil \frac{(2^{L'}-MQ)^{M-1}}{(M-1)!} \rceil\right]$. Let us assume for now that the mapping F_S^H is given.

For a set $S \in \binom{\{0,1\}^{L'}}{\leq M}$, define the characteristic vector $\mathbb{1}(S) \in \{0,1\}^{2^{L'}}$ of S by

$$\mathbb{1}(S)_i = \begin{cases} 1 & \text{if the binary presentation of } i \text{ is in } S \\ 0 & \text{else} \end{cases}.$$

Notice that the Hamming weight of $\mathbb{1}(S)$ is M for every $S \in \binom{\{0,1\}^{L'}}{M}$. The following lemma states that k substitution errors in a set of strings S result in at most $2k$ bit flips in $\mathbb{1}(S)$.

Lemma 6.3.2. *For $S_1, S_2 \in \binom{\{0,1\}^{L'}}{\leq M}$, if $S_1 \in \mathcal{B}_k^H(S_2)$, then $d_H(\mathbb{1}(S_1), \mathbb{1}(S_2)) \leq 2k$, where $d_H(\mathbb{1}(S_1), \mathbb{1}(S_2))$ is the Hamming distance between $\mathbb{1}(S_1)$ and $\mathbb{1}(S_2)$.*

Proof. Note that $|S_1 \backslash S_2| \leq k$ and $|S_2 \backslash S_1| \leq k$. Hence $d_H(\mathbb{1}(S_1), \mathbb{1}(S_2)) = |S_1 \backslash S_2 \cup S_2 \backslash S_1| \leq 2k$. □

We are ready to present the code construction. We use a set $S \in \mathcal{S}^H$ as indexing bits and protect the vector $\mathbb{1}_S$ from substitution errors. Note that any two strings in the set S have Hamming distance at least $2k+1$. Hence, knowing the set S, each string of indexing bits can be extracted from its erroneous version using a minimum distance decoder, which finds the unique string in S that is within Hamming distance k from it. The details are given as follows.

Consider the data $\mathbf{d} \in D$ to be encoded as a tuple $\mathbf{d} = (d_1, \mathbf{d}_2)$, where $d_1 \in [\lceil \frac{(2^{L'}-MQ)^{M-1}}{(M-1)!} \rceil]$ and

$$\mathbf{d}_2 \in \{0,1\}^{M(L-L')-4kL'-2k\lceil \log ML \rceil}.$$

Given $(d_1, \mathbf{d}_2)$, the codeword $\{\mathbf{x}_i\}_{i=1}^M$ is generated by the following procedure.

Encoding:

(1) Let $F_S^H(d_1) = \{\mathbf{a}_1, \ldots, \mathbf{a}_M\} \in \mathcal{S}^H$ such that $\mathbf{a}_1 = \mathbb{1}_{L'}$ and the $\mathbf{a}_i$'s are sorted in a descending lexicographic order. Let $(x_{i,1}, \ldots, x_{i,L'}) = \mathbf{a}_i$, for $i \in [M]$.

(2) Let $(x_{1,L'+1}, \ldots, x_{1,L'+4kL'}) = RS_{2k}(\mathbb{1}(\{\mathbf{a}_1, \ldots, \mathbf{a}_M\}))$, where $RS_{2k}(\mathbb{1}(\{\mathbf{a}_1, \ldots, \mathbf{a}_M\}))$ is the redundancy of a systematic Reed-Solomon code that corrects $2k$ substitutions in $\mathbb{1}(\{\mathbf{a}_1, \ldots, \mathbf{a}_M\})$.

(3) Place the information bits of $\mathbf{d}_2$ in bits

$$(x_{1,L'+4kL'+1}, \ldots, x_{1,L}),$$

$$(x_{M,L'+1},\dots,x_{M,L-2k\lceil\log ML\rceil}); \text{ and}$$
$$(x_{i,L'+1},\dots,x_{i,L}) \text{ for } i\in[2,M-1].$$

(4) Define

$$\mathbf{m}=(\mathbf{x}_1,\dots,\mathbf{x}_{M-1},(x_{M,1},\dots,x_{M,L-2k\lceil\log ML\rceil}))$$

and let $(x_{M,L-2k\lceil\log ML\rceil+1},\dots,x_{M,L})=RS_k(\mathbf{m})$, where $RS_k(\mathbf{m})$ is the Reed-Solomon redundancy that corrects k substitution errors in $\mathbf{m}$. Note that $(\mathbf{x}_1,\dots,\mathbf{x}_M)=(\mathbf{m},RS_k(\mathbf{m}))$ is a k-substitution correcting Reed-Solomon code.

Upon receiving the erroneous version[1] $\{\mathbf{x}'_1,\dots,\mathbf{x}'_M\}$, the decoding procedure is as follows.

Decoding:

(1) Note that during the encoding process, the redundancy bits that correct the vector $\mathbb{1}(\{\mathbf{a}_i\}_{i=1}^M)$, i.e., the characteristic vector of the set of indexing bits $\{(x_{i,1},\dots,x_{i,L'})\}_{i=1}^M$, are stored in $\mathbf{x}_1$. Hence we must first identify the erroneous copy of $\mathbf{x}_1$. To this end, find the unique string $\mathbf{x}'_{i_0}$ such that $(x'_{i_0,1},\dots,x'_{i_0,L'})$ has at least $L'-k$ many 1-entries. Since the strings $\{\mathbf{x}_i\}_{i=1}^M$ have Hamming distance at least $2k+1$, there is a unique such string, which is the erroneous copy of $\{(x_{1,1},\dots,x_{1,L'})\}_{i=1}^M$. Hence $\mathbf{x}'_{i_0}$ is an erroneous copy of $\mathbf{x}_1$ and the string

$$(x'_{i_0,L'+1},\dots,x'_{i_0,L'+4kL})$$

is an erroneous copy of $(x_{1,L'+1},\dots,x_{1,L'+4kL'})=RS_{2k}(\mathbb{1}(\{\mathbf{a}_i\}_{i=1}^M))$.

(2) Let k be the number of substitution errors that occur to the indexing bits $\{x_{i,1},\dots,x_{i,L'}\}_{i=1}^M$. According to Lemma 6.3.2, the vector $\mathbb{1}(\{(x_{i,1},\dots,x_{i,L'})\}_{i=1}^M)$ is within Hamming distance $2k$ from the vector $\mathbb{1}(\{(x'_{i,1},\dots,x'_{i,L'})\}_{i=1}^M)$. Hence the Hamming distance between

$$\mathbf{s}_1=(\mathbb{1}(\{(x'_{i,1},\dots,x'_{i,L'})\}_{i=1}^M),(x'_{i_0,L'+1},\dots,x'_{i_0,L'+4kL})) \text{ and}$$
$$\mathbf{s}_2=(\mathbb{1}(\{\mathbf{a}_i\}_{i=1}^M),RS_{2k}(\mathbb{1}(\{\mathbf{a}_i\}_{i=1}^M)))$$

is at most $2k$. Since $\mathbf{s}_2$ is a $2k$ error correcting Reed Solomon code, it can be recovered from $\mathbf{s}_1$ using the Reed-Solomon decoder. Recover $d_1=(F_S^H)^{-1}(\{\mathbf{a}_i\}_{i=1}^M)$.

[1]Since the strings $\{\mathbf{x}_i\}_{i=1}^M$ have distance at least $2k+1$ with each other, the strings $\{\mathbf{x}'_i\}_{i=1}^M$ are different.

(3) Since $\mathbf{s}_2$ is recovered, the strings $\{(x_{i,1},\ldots,x_{i,L'})\}_{i=1}^{M} = \{\mathbf{a}_i\}_{i=1}^{M}$ are known. Sort $\{(x_{i,1},\ldots,x_{i,L'})\}_{i=1}^{M}$ lexicographically in descending order. For each $i \in [M]$, find the unique $\pi(i) \in [M]$ such that $d_H((x'_{\pi(i),1},\ldots,x'_{\pi(i),L'}),$ $(x_{i,1},\ldots,x_{i,L'})) \leq k$ (note that $i_0 = \pi(1)$). Similar to Step (1), we conclude that the string $\mathbf{x}'_{\pi(i)}$ is an erroneous copy of $\mathbf{x}_i$, $i \in [M]$, since the Hamming distance between $\mathbf{x}_j$ and $\mathbf{x}_i$ is at least $2k+1$ for $j \neq i$. Hence, the identify of $\{(x_{i,1},\ldots,x_{i,L'})\}_{i=1}^{M}$ are determined from $\{(x'_{i,1},\ldots,x'_{i,L'})\}_{i=1}^{M}$.

(4) Since $\mathbf{x}'_{\pi(i)}$ is an erroneous copy of $\mathbf{x}_i$, $i \in [M]$, it follows that the concatenation $\mathbf{s}' = (\mathbf{x}'_{\pi(1)},\ldots,\mathbf{x}'_{\pi(M)})$ is an erroneous copy of $(\mathbf{x}_1,\ldots,\mathbf{x}_M) = (\mathbf{m}, RS_k(\mathbf{m}))$, where $\mathbf{m}$ is defined in Step (4) in the encoding procedure. Therefore, $(\mathbf{x}_1,\ldots,\mathbf{x}_M)$ and thus $\mathbf{d}_2$ can be recovered from $(\mathbf{x}'_{\pi(1)},\ldots,\mathbf{x}'_{\pi(M)})$ by using the Reed-Solomon decoder.

(5) Output $(d_1, \mathbf{d}_2)$.

Therefore, the codeword $\{\mathbf{x}_i\}_{i=1}^{M}$ can be recovered. The redundancy of the code is

$$r(C) = \log\binom{2^L}{M} - \log\lceil\frac{(2^{L'}-MQ)^{M-1}}{(M-1)!}\rceil - [M(L-L') - 4kL' - 2k\lceil\log ML\rceil] \tag{6.2}$$

$$\overset{(a)}{\leq} 2k\log ML + (12k+2)\log M + O(k^3) + O(k\log\log ML), \tag{6.3}$$

where (a) will be proved in Appendix 6.7. The complexity of the encoding/decoding is that of computing the function F_S^H, which as will be discussed in Sec. 6.4, is $poly(M, L, k)$.

6.4 Computing F_S^H in Polynomial Time

In this section we present a polynomial time algorithm to compute the function F_S^H and thus complete the code construction in Sec. 6.3. The result is as follows.

Theorem 6.4.1. *For integers M, L, k, $L' \triangleq 3\log M + 4k^2 + 1$, and $Q = \sum_{i=0}^{2k}\binom{L'}{i}$, there exists an invertible mapping $F_S^H : \left[\binom{2^{L'}-(M-1)Q+M-1}{M-1}\right] \to \binom{\{0,1\}^{L'}}{M}$, computable in $poly(M, L)$ time, such that for any $d \in [\lceil\frac{(2^{L'}-MQ)^{M-1}}{(M-1)!}\rceil]$, we have that $F_S^H(d) \in \mathcal{S}^H$.*

The algorithm has a greedy flavor in the sense that the strings $\mathbf{a}_1,\ldots,\mathbf{a}_M$ are generated sequentially and each string $\mathbf{a}_i$, $i \in [2, M]$ is generated bit by bit. The algorithm consists of two steps. In the first step we map the integer $d \in [\lceil\frac{(2^{L'}-MQ)^{M-1}}{(M-1)!}\rceil]$ into $M-1$ integers $q_1, \ldots, q_M \in [2^{L'}]$ such that $q_1 = 2^{L'}$ and $q_{i+1} \le q_i - Q$ for $i \in [M-1]$. In the second step, we use q_i to generate $\mathbf{a}_i$ sequentially for $i \in [2, M]$. The first step is given in the following lemma.

Lemma 6.4.1. *There exists an invertible map* $F_Q^H : [\lceil\frac{(2^{L'}-MQ)^{M-1}}{(M-1)!}\rceil] \to \binom{[2^{L'}]}{M}$, *computable in* $poly(L', M)$ *time, that maps and integer* $d \in [\lceil\frac{(2^{L'}-MQ)^{M-1}}{(M-1)!}\rceil]$ *to an integer tuple* $(q_1, \ldots, q_M)$ *such that* $q_1 = 2^{L'}$ *and* $q_i \ge q_{i+1} + Q$ *for* $i \in [M-1]$.

Proof. Recall the *combinatorial numbering map* F_{com} that maps an integer in the range $[\binom{n}{m}]$ to a set of m different and unordered integers in the range $[n]$ for integers n and $m \le n$. Since $\frac{(2^{L'}-MQ)^{M-1}}{(M-1)!} \le \binom{2^{L'}-MQ+M-1}{M-1}$, we can map $d \in [\lceil\frac{(2^{L'}-MQ)^{M-1}}{(M-1)!}\rceil]$ to $M-1$ integers $F_{com}(d) = \{q'_2, \ldots, q'_M\}$ such that $2^{L'} - MQ + M - 1 \ge q'_2 > q'_3 > \ldots > q'_M$. Let $q_1 = 2^{L'}$, $q_i = q'_i + (M-i+1)(Q-1)$ for $i \in [2, M]$, and $F_Q^H(d) = \{q_1, \ldots, q_M\}$. Then we have that $q_2 \le 2^{L'} - Q$ and that $q_i \ge q_{i+1} + Q$ for $i \in [2, M-1]$. Since the map F_{com} is invertible and computed in $poly(L', M)$ time, so is the map F_Q^H. □

We now turn to the second step. Given the integers $F_Q^H(d) = (q_1, \ldots, q_M)$, we generate the indexing bits $\{\mathbf{a}_i = (x_{i,1}, \ldots, x_{i,L'})\}_{i=1}^M \in \mathcal{S}^H$. First, we have that $\mathbf{a}_1 = \mathbb{1}_{L'}$. The algorithm generates the indexing string $\mathbf{a}_i$ sequentially for $i \in [2, M]$. Each indexing string $\mathbf{a}_i$ is generated bit by bit in a recursive manner. We first give the following definition, on which the algorithm is based.

For a set of strings $A \subset \{0,1\}^{L'}$ and a string $\mathbf{a} \in \{0,1\}^\ell$ of length $\ell \in [L']$. Denote

$$N_H(\mathbf{a}, A) = \sum_{\mathbf{c}:\mathbf{c}\in A} |\{\mathbf{c}' : (c'_1, \ldots, c'_\ell) = \mathbf{a} \text{ and } d_H(\mathbf{c}', \mathbf{c}) \le 2k\}|$$

as the sum of the number of sequences that have prefix $\mathbf{a}$ and have Hamming distance at most $2k$ from $\mathbf{c}$ over $\mathbf{c} \in A$. The number $N_H(\mathbf{a}, A)$ has the following properties that will be useful in our proof. The first property implies that

$$2^{L'-\ell} - N_H(\mathbf{a}, A) = (2^{L'-\ell-1} - N_H((\mathbf{a}, 0), A)) + (2^{L'-\ell-1} - N_H((\mathbf{a}, 1), A)), \quad (6.4)$$

which enables a recursion to generate each sequence $\mathbf{a}_i$. The second property provides a way to compute $N_H(\mathbf{a}, A)$.

Lemma 6.4.2. *1. For any sequence* $\mathbf{a} \in \{0,1\}^\ell$ *of length* $\ell \in [L'-1]$ *and set* $A \subset \{0,1\}^{L'}$, *we have*

$$N_H(\mathbf{a}, A) = N_H((\mathbf{a}, 0), A) + N_H((\mathbf{a}, 1), A), \quad (6.5)$$

where $(\mathbf{a}, 0)$ *or* $(\mathbf{a}, 1)$ *is the concatenation of* $\mathbf{a}$ *and a* 0 *or* 1 *bit respectively.*

2. *For any* $\mathbf{a} \in \{0,1\}^\ell$ *and* $A \subset \{0,1\}^{L'}$*, we have*

$$N_H(\mathbf{a}, A) = \sum_{\mathbf{c}:\mathbf{c}\in A} \sum_{i=0}^{2k-d_H(\mathbf{a},(c_1,\ldots,c_\ell))} \binom{L'-\ell}{i}. \tag{6.6}$$

Proof. Note that for any sequence $\mathbf{c}$, the $\ell+1$-th bit of any sequence $\mathbf{c}'$ satisfying $(c'_1,\ldots,c'_\ell) = \mathbf{a}$ is either 0 or 1. Hence

$$\begin{aligned}
&|\{\mathbf{c}' : (c'_1,\ldots,c'_\ell) = \mathbf{a} \text{ and } d_H(\mathbf{c}',\mathbf{c}) \leq 2k\}| \\
=&|\{\mathbf{c}' : (c'_1,\ldots,c'_{\ell+1}) = (\mathbf{a},0) \text{ and } d_H(\mathbf{c}',\mathbf{c}) \leq 2k\}| \\
&+ |\{\mathbf{c}' : (c'_1,\ldots,c'_{\ell+1}) = (\mathbf{a},1) \text{ and } d_H(\mathbf{c}',\mathbf{c}) \leq 2k\}|,
\end{aligned}$$

which implies Eq. (6.5). Moreover, for any sequence $\mathbf{c} \in \{0,1\}^{L'}$, we have that

$$|\{\mathbf{c}' : (c'_1,\ldots,c'_\ell) = \mathbf{a} \text{ and } d_H(\mathbf{c}',\mathbf{c}) \leq 2k\}| = \sum_{i=0}^{2k-d_H(\mathbf{a},(c_1,\ldots,c_\ell))} \binom{L'-\ell}{i}.$$

Hence the number $N_H(\mathbf{a}, A)$ can be computed by Eq. (6.6). □

Next, we present the algorithm that takes $F_Q^H(d) = (q_1,\ldots,q_M)$ as input and outputs $\mathbf{a}_i$ such that $\{\mathbf{a}_1,\ldots,\mathbf{a}_M\} \in \mathcal{S}^H$ and that the decimal presentation decimal$(\mathbf{a}_i)$ of $\mathbf{a}_i, i \in [M]$ satisfies

$$\text{decimal}(\mathbf{a}_i) = q_i - 1 + \sum_{\ell:a_{i,\ell}=1 \text{ and } \ell\in[L']} N_H((a_{i,1},\ldots,a_{i,\ell-1},0),\{\mathbf{a}_j\}_{j=1}^{i-1}). \tag{6.7}$$

We then show that the sequences $\mathbf{a}_i, i \in [M]$ satisfying (6.7) are decodable, i.e., we can recover the tuple $(q_1,\ldots,q_M)$ from $\{\mathbf{a}_1,\ldots,\mathbf{a}_M\}$.

Encoding:

```
for $i \in [M]$, do
$q = q_i$.
  for $\ell \in [L']$, do
    if $2^{L'-\ell} - N_H((a_{i,1},\ldots,a_{i,\ell-1},0),\{\mathbf{a}_j\}_{j=1}^{i-1}) \geq q$,
      then $a_{i,\ell} = 0$.
```

```
        else
            $q = q - (2^{L'-\ell} - N_H((a_{i,1}, \dots, a_{i,\ell-1}, 0), \{\mathbf{a}_j\}_{j=1}^{i-1}))$,
            $a_{i,\ell} = 1$.
        end if
    end for
end for
return $\{\mathbf{a}_1, \dots, \mathbf{a}_M\}$.
```

The generation of $\mathbf{a}_i, i \in [M]$ in the encoding procedure can be intuitively characterized as walking on a complete binary tree of $L' + 1$ layers. The walk starts at layer 1, i.e., the root of the binary tree, and ends at layer $L' + 1$ at one of the leaf nodes. At each step, it goes to one of its two child nodes, which represent the bits 0 and 1 respectively. Each string $\mathbf{a}_i, i \in [M]$ is represented by the path of a walk. For each path $\mathbf{a}_i = (a_{i,1}, \dots, a_{i,L'})$ and each layer $\ell \in [L']$, assign the weight $w(a_{i,\ell}) = 2^{L'-\ell} - N_H((a_{i,1}, \dots, a_{i,\ell}), \{\mathbf{a}_j\}_{j=1}^{i-1})$ to node $a_{i,\ell}$ in the ℓ-th layer, and the weight $w(\bar{a}_{i,\ell}) = 2^{L'-\ell} - N_H((a_{i,1}, \dots, 1 - a_{i,\ell}), \{\mathbf{a}_j\}_{j=1}^{i-1})$ to the brother node of node $a_{i,\ell}$, i.e., the node that shares the same parent node with $a_{i,\ell}$. From Eq. (6.5) we have that $w(a_{i,\ell}) = w(a_{i,\ell+1}) + w(\bar{a}_{i,\ell+1})$ for $\ell \in [L' - 1]$. Moreover, we have that $0 < q \leq w(a_{i,\ell})$ after the ℓ-th inner for loop in the i-th outer for loop. This is formalized in the following lemma, which can be used to prove that Eq. (6.7) holds and that $\{\mathbf{a}_1, \dots, \mathbf{a}_M\} \in \mathcal{S}^H$.

Lemma 6.4.3. *After the ℓ-th, $\ell \in [L']$, inner for loop in the i-th, $i \in [M]$, outer for loop in the encoding procedure, we have that*

$$0 < q \leq 2^{L'-\ell} - N_H((a_{i,1}, \dots, a_{i,\ell}), \{\mathbf{a}_j\}_{j=1}^{i-1}). \tag{6.8}$$

At the end of the i-th outer for loop, we have that $q = 1$.

Proof. We prove Eq. (6.8) by induction on ℓ. For $\ell = 1$, according to Lemma 6.4.1, we have $0 < q = q_i \leq 2^{L'} - (i - 1)Q$ at the beginning of the i-th outer for loop. If $a_{i,1} = 0$, then according to the if condition in the encoding procedure, we have that $0 < q \leq 2^{L'-\ell} - N_H(0, \{\mathbf{a}_j\}_{j=1}^{i-1})$ for $\ell = 1$, which proves (6.8). Otherwise if $a_{i,1} = 1$, we have

$$0 < q = q_i - (2^{L'-\ell} - N_H(0, \{\mathbf{a}_j\}_{j=1}^{i-1}))$$

$$\leq 2^{L'} - (i-1)Q - (2^{L'-\ell} - N_H(0, \{\mathbf{a}_j\}_{j=1}^{i-1}))$$
$$\overset{(a)}{=} (2^{L'-1} - N_H(1, \{\mathbf{a}_j\}_{j=1}^{i-1})),$$

where (a) holds since by definition of $N_H(\mathbf{a}, A)$, we have that

$$\begin{aligned} N_H(0, \{\mathbf{a}_j\}_{j=1}^{i-1}) + N_H(1, \{\mathbf{a}_j\}_{j=1}^{i-1}) &= \sum_{j=1}^{i-1} |\{\mathbf{c} : d_H(\mathbf{c}, \mathbf{a}_j) \leq 2k\}| \\ &= \sum_{j=1}^{i-1} Q \\ &= (i-1)Q. \end{aligned}$$

Hence the claim holds for $\ell = 1$. Suppose Eq. (6.8) holds for $\ell = m$. For $\ell = m+1$, if $a_{i,m+1} = 0$, then from Step (3), we have $0 < q \leq 2^{L'-m-1} - N_H((a_{i,1}, \ldots, a_{i,m}, 0), \{\mathbf{a}_j\}_{j=1}^{i-1})$. Otherwise if $a_{i,m+1} = 1$, we have that

$$\begin{aligned} 0 < q = q_i - (2^{L'-m-1} - N_H((a_{i,1}, \ldots, a_{i,\ell}, 0), \{\mathbf{a}_j\}_{j=1}^{i-1})) \\ \leq 2^{L'-m} - N_H((a_{i,1}, \ldots, a_{i,m}), \{\mathbf{a}_j\}_{j=1}^{i-1}) \\ - (2^{L'-m-1} - N_H((a_{i,1}, \ldots, a_{i,m}, 0), \{\mathbf{a}_j\}_{j=1}^{i-1})) \\ \overset{(b)}{=} (2^{L'-m-1} - N_H((a_{i,1}, \ldots, a_{i,m}, 1), \{\mathbf{a}_j\}_{j=1}^{i-1})), \end{aligned}$$

where (b) follows from Eq. (6.5). Therefore, Eq. (6.8) holds for $\ell = m+1$ and thus holds for $\ell \in [L']$. Hence at the end of Step (2) we have that

$$0 < q \leq 2^{L'-L'} - N_H(\mathbf{a}_i, \{\mathbf{a}_j\}_{j=1}^{i-1}) \leq 1. \tag{6.9}$$

Hence q equals 1 at the end of Step (2). □

We now show that the strings $\{\mathbf{a}_1, \ldots, \mathbf{a}_M\}$ generated in the encoding procedure belong to $\mathcal{S}_H$. By Lemma 6.4.3, we have

$$q = 2^{L'-L'} - N_H(\mathbf{a}_i, \{\mathbf{a}_j\}_{j=1}^{i-1}) = 1,$$

at the end of each round of Step (2) in the encoding procedure. This implies that $N_H(\mathbf{a}_i, \{\mathbf{a}_j\}_{j=1}^{i-1}) = 0$ and thus $d_H(\mathbf{a}_i, \mathbf{a}_j) \geq 2k+1$ for $i \in [2, M]$ and $j \in [i-1]$. Moreover, since $q_1 = 2^{L'}$, we have that $\mathbf{a}_1 = \mathbb{1}_{L'}$. Therefore, $\{\mathbf{a}_i\}_{i=1}^{M} \in \mathcal{S}_H$.

Next, we use Lemma 6.4.3 to show that the strings $\{\mathbf{a}_i\}_{i=1}^{M}$ satisfy Eq. (6.7).

Lemma 6.4.4. *The output $\{\mathbf{a}_i\}_{i=1}^{M}$ of the encoding algorithm satisfies Eq.* (6.7).

Proof. Note that in each inner for loop, the number q is subtracted by $2^{L'-\ell} - N_H((a_{i,1},\ldots,a_{i,\ell-1},0),\{\mathbf{a}_j\}_{j=1}^{i-1})$ only when $a_{i,\ell} = 1$ and $\ell \in [L']$. Since the number q equals q_i at the beginning of each outer for loop, and from Lemma 6.4.3 equals 1 at the end of each outer for loop, hence we have that

$$q_i - \sum_{\ell : a_{i,\ell}=1 \text{ and } \ell\in[L']} (2^{L'-\ell} - N_H((a_{i,1},\ldots,a_{i,\ell-1},0),\{\mathbf{a}_j\}_{j=1}^{i-1})) = 1,$$

which implies (6.7). □

Remark 6.4.1. *By definition of $N_H(\mathbf{a}, A)$, we have the following alternative characterization of* decimal$(\mathbf{a}_i)$*, $i \in [M]$.*

$$\text{decimal}(\mathbf{a}_i) = q_i - 1 + \sum_{j=1}^{i-1} |\{\mathbf{c} : \mathit{decimal}(\mathbf{c}) < \mathit{decimal}(\mathbf{a}_i) \text{ and } d_H(\mathbf{c},\mathbf{a}_j) \le 2k\}|, \tag{6.10}$$

which is $q_i - 1$ plus the sum of number of strings that are lexicographically less than $\mathbf{a}_i$ and have Hamming distance at most $2k$ from $\mathbf{a}_j$ over $j < i$.

Lemma 6.4.4 immediately implies a decoding algorithm that transforms $\{\mathbf{a}_i\}_{i=1}^M$ back to $(q_1,\ldots,q_M)$.

Decoding:

(1) Order the strings $\{\mathbf{a}_i\}_{i=1}^M$ such that $\mathbf{a}_1 > \mathbf{a}_2 > \ldots > \mathbf{a}_M$.

(2) For $i \in [M]$,

$$q_i = \text{decimal}(\mathbf{a}_i) + 1 + \sum_{\ell : a_{i,\ell}=1 \text{ and } \ell\in[L']} N_H((a_{i,1},\ldots,a_{i,\ell-1},0),\{\mathbf{a}_j\}_{j=1}^{i-1}). \tag{6.11}$$

To show that the decoding is correct, we prove that the string $\mathbf{a}_i$, $i \in [M]$ generated in the encoding procedure satisfies

$$\mathbf{a}_1 > \mathbf{a}_2 > \ldots > \mathbf{a}_M. \tag{6.12}$$

Then we conclude that the string $\mathbf{a}_i$ obtained by ordering $\{\mathbf{a}_i\}_{i=1}^M$ in Step (1) in the decoding procedure satisfies Eq. (6.7). Hence we have Eq. (6.23) and thus q_i, $i \in [M]$ can be recovered. Suppose on the contrary, there exist $\mathbf{a}_{i_1} > \mathbf{a}_{i_2}$ for some $i_1 > i_2$.

Let ℓ^* be the most significant bit where $\mathbf{a}_{i_1}$ and $\mathbf{a}_{i_2}$ differ, i.e., $(a_{i_1,1},\ldots,a_{i_1,\ell^*-1}) = (a_{i_2,1},\ldots,a_{i_2,\ell^*-1})$ and $a_{i_1,\ell^*}=1$ and $a_{i_2,\ell^*}=0$. Then according to the if statement in the encoding procedure, we have that

$$q_{i_1} - \sum_{\ell:a_{i_1,\ell}=1 \text{ and } \ell\in[\ell^*]} (2^{L'-\ell} - N_H((a_{i_1,1},\ldots,a_{i_1,\ell-1},0),\{\mathbf{a}_j\}_{j=1}^{i_1-1})) > 0 \text{ and}$$
$$q_{i_2} - \sum_{\ell:a_{i_1,\ell}=1 \text{ and } \ell\in[\ell*]} (2^{L'-\ell} - N_H((a_{i_1,1},\ldots,a_{i_1,\ell-1},0),\{\mathbf{a}_j\}_{j=1}^{i_2-1})) \leq 0,$$

which implies that

$$\begin{aligned}
q_{i_2} - q_{i_1} < & \sum_{\ell:a_{i_1,\ell}=1 \text{ and } \ell\in[\ell^*]} (2^{L'-\ell} - N_H((a_{i_1,1},\ldots,a_{i_1,\ell-1},0),\{\mathbf{a}_j\}_{j=1}^{i_2-1})) \\
& - \sum_{\ell:a_{i_1,\ell}=1 \text{ and } \ell\in[\ell^*]} (2^{L'-\ell} - N_H((a_{i_1,1},\ldots,a_{i_1,\ell-1},0),\{\mathbf{a}_j\}_{j=1}^{i_1-1})) \\
= & \sum_{\ell:a_{i_1,\ell}=1 \text{ and } \ell\in[\ell^*]} (N_H((a_{i_1,1},\ldots,a_{i_1,\ell-1},0),\{\mathbf{a}_j\}_{j=1}^{i_1-1}) \\
& - N_H((a_{i_1,1},\ldots,a_{i_1,\ell-1},0),\{\mathbf{a}_j\}_{j=1}^{i_2-1})) \\
= & \sum_{\ell:a_{i_1,\ell}=1 \text{ and } \ell\in[\ell^*]} N_H((a_{i_1,1},\ldots,a_{i_1,\ell-1},0),\{\mathbf{a}_j\}_{j=i_2}^{i_1-1}) \\
\overset{(a)}{\leq} & \sum_{j=i_2}^{i_1-1} |\mathbf{c} : d_H(\mathbf{c},\mathbf{a}_j) \leq 2k| \\
= & (i_1 - i_2)Q,
\end{aligned} \tag{6.13}$$

where (a) follows from the definition of $N_H(\mathbf{a},A)$ and the fact that the strings which have $(a_{i_1,1},\ldots,a_{i_1,\ell_1-1},0)$ and $(a_{i_1,1},\ldots,a_{i_1,\ell_2-1},0)$ as prefixes, respectively, where $a_{i_1,\ell_1=1}$, $a_{i_1,\ell_2=1}$ and $\ell_1 \neq \ell_2$, are different. Eq. (6.13) contradicts to the fact that the integers $(q_1,\ldots,q_M) = F_Q^H(d)$ satisfy $q_i - q_{i+1} > Q$ for $i \in [M-1]$, which implies $q_{i_1} - q_{i_2} \geq (i_1 - i_2)Q$.

Since the calculation of $N_H(\mathbf{a},A)$ has polynomial complexity, the complexity of the encoding/decoding procedure is polynomial in M and L'.

6.5 Robust Indexing for Deletion/Insertion Errors

In this section we show how the idea of robust indexing can be used for correcting deletion/insertion errors over sliced channels. The redundancy of the construction is $O(k \log ML)$ for constant k.

Theorem 6.5.1. *For integers M, L, k, and $L' \triangleq 3\log M + 4k^2 + 1$. If $L' + 4kL' + 2k\log(4kL') \leq L$, there exists a k-deletion code, computable in $poly(M, L)$ time, that has redundancy $8k\log ML + (12k+2)\log M + O(k^3) + o(\log ML)$.*

Similar to the construction in Sec. 6.3, we use the first L' bits $(x_{i,1}, \dots, x_{i,L'})$, $i \in [M]$ in each string $\mathbf{x}_i$ as indexing bits and sort the strings $\{\mathbf{x}_i\}$ according to the lexicographic order of $\{(x_{i,1}, \dots, x_{i,L'})\}_{i=1}^M$. To protect the ordering, we use Reed-Solomon code to protect the characteristic vector $\mathbb{1}(\{(x_{i,1}, \dots, x_{i,L'})\}_{i=1}^M)$. The difference is that in this section, we construct the indexing bits $\{(x_{i,1}, \dots, x_{i,L'})\}_{i=1}^M$ such that the mutual deletion distance among $\{(x_{i,1}, \dots, x_{i,L'})\}_{i=1}^M$, rather than the mutual Hamming distance considered in Sec. 6.3, is at least $2k+1$, i.e., the deletion balls $\mathcal{D}_k((x_{i,1}, \dots, x_{i,L'}))$ and $\mathcal{D}_k((x_{j,1}, \dots, x_{j,L'}))$ do not intersect for $i \neq j$, where the deletion ball $\mathcal{D}_k(\mathbf{u})$ of a string $\mathbf{u}\{0,1\}^n$ is the set of all length $n-k$ subsequence of $\mathbf{u}$. Define

$$\mathcal{S}^D = \left\{\{\mathbf{a}_1, \dots, \mathbf{a}_M\} : \mathcal{D}_k(\mathbf{a}_i) \cap \mathcal{D}_k(\mathbf{a}_j) = \emptyset \text{ for } i \neq j\right\}.$$

The construction is based on the following two lemmas, where the first one is robust indexing for deletion/insertion errors, which will be proved in Sec. 6.5 and the second one is a deletion code construction, which we presented in Ch. 4.

Lemma 6.5.1. *For $P = 2^k \binom{L'}{k}^2$, there exists an invertible mapping*

$$F_S^D : \left[\binom{2^{L'} - (M-1)P + M - 1}{M-1}\right] \to \binom{\{0,1\}^{L'}}{M},$$

computable in $poly(M, L)$ time, such that for any $d \in [\lceil \frac{(2^{L'} - MP)^{M-1}}{(M-1)!} \rceil]$, we have that $F_S^D(d) \in \mathcal{S}^D$.

Lemma 6.5.2. *(Corollary of Theorem 4.1.2) For any integer n and $N = n + 4k\log n + o(\log n)$, there exists a systematic encoding function $Enc : \{0,1\}^n \to \{0,1\}^N$, computed in $O(n^{2k+1})$ time, and a decoding function $Dec : \{0,1\}^{N-k} \to \{0,1\}^n$, computed in $O(n^{k+1})$ time, such that for any $\mathbf{c} \in \{0,1\}^n$ and substring $\mathbf{d} \in \{0,1\}^{N-k}$ of $Enc(\mathbf{c})$, we have that $Dec(\mathbf{d}) = \mathbf{c}$.*

Code Constructions

The code construction is the same as that in Sec. 6.3 except that here, the indexing bits $\{(x_{i,1}, \dots, x_{i,L'})\}_{i=1}^M$ are generated using the map F_S^D. In addition, a deletion code in Lemma 6.5.2 is used to protect the concatenated string.

Let the data $\mathbf{d} \in D$ to be encoded be a tuple $\mathbf{d} = (d_1, \mathbf{d}_2)$, where $d_1 \in [\lceil \frac{(2^{L'}-MP)^{M-1}}{(M-1)!} \rceil]$ and

$$\mathbf{d}_2 \in \{0,1\}^n$$

such that $n + 4k\log n + o(\log n) = M(L-L') - 4kL'$, which implies that $n = M(L-L') - 4kL' - 4k\lceil \log ML \rceil - o(\log ML)$. We briefly present the encoding/decoding procedure as follows.

Encoding:

(1) Let $F_S^D(d_1) = \{\mathbf{a}_1, \ldots, \mathbf{a}_M\} \in \mathcal{S}^H$ such that $\mathbf{a}_1 = \mathbb{1}_{L'}$ and $\mathbf{a}_1 > \mathbf{a}_2 > \ldots > \mathbf{a}_M$. Let $(x_{i,1}, \ldots, x_{i,L'}) = \mathbf{a}_i$, for $i \in [M]$.

(2) Let

$$\begin{aligned}&(x_{1,L'+1}, \ldots, x_{1,L'+4kL'+4k\log(4kL')+o(\log(4kL'))})\\ =&Enc(RS_{2k}(\mathbb{1}(\{\mathbf{a}_1, \ldots, \mathbf{a}_M\})))\end{aligned}$$

(3) Place the deletion code $Enc(\mathbf{d}_2)$ in bits

$$\begin{aligned}&(x_{1,L'+4kL'+4k\log(4kL')+o(\log(4kL'))+1}, \ldots, x_{1,L}), \text{ and}\\ &(x_{i,L'+1}, \ldots, x_{i,L}) \text{ for } i \in [2, M].\end{aligned}$$

Upon receiving $\{\mathbf{x}'_i\}_{i=1}^M$, the decoding procedure is as follows.

Decoding:

(1) Find the unique string $\mathbf{x}'_{i_0}$ such that $(x'_{i_0,1}, \ldots, x'_{i_0,L'-k}) = \mathbb{1}_{L'-k}$. Then $\mathbf{x}'_{i_0}$ is an erroneous copy of $\mathbf{x}_1$ and the string

$$(x'_{i_0,L'+1}, \ldots, x'_{i_0,L'+4kL+4k\log(4kL')+o(\log(4kL'))-k})$$

is an erroneous copy of

$$(x_{1,L'+1}, \ldots, x_{1,L'+4kL'+4k\log(4kL')+o(\log(4kL'))}) = Enc(RS_{2k}(\mathbb{1}(\{\mathbf{a}_i\}_{i=1}^M))).$$

Correct the vector $RS_{2k}(\mathbb{1}(\{\mathbf{a}_i\}_{i=1}^M))$ and use it to recover $\mathbb{1}(\{\mathbf{a}_i\}_{i=1}^M)$, and thus the indexing bits $\{(x_{i,1}, \ldots, x_{i,L'})\}_{i=1}^M$. Recover $d_1 = (F_S^H)^{-1}(\{\mathbf{a}_i\}_{i=1}^M)$.

(2) For each $i \in [M]$, find the unique $\pi(i) \in [M]$ such that $(x'_{\pi(i),1}, \ldots, x'_{\pi(i),L'-k})$ is a length $L'-k$ substring of $(x_{i,1}, \ldots, x_{i,L'})$ (note that $\pi(1) = i_0$). Checking if a string is a substring of another can be done in linear time using a greedy algorithm.

(4) Since $\mathbf{x}'_{\pi(i)}$ is an erroneous copy of $\mathbf{x}_i$, $i \in [M]$, the concatenation $\mathbf{m}' = ((x'_{\pi(1),L'+4kL'+4k\log(4kL')+o(\log(4kL'))+1}, \ldots, x'_{\pi(1),L_1}), \mathbf{x}'_{\pi(2)}, \ldots, \mathbf{x}'_{\pi(M)})$, where L_1 is the length of $\mathbf{x}'_{\pi(1)}$, is an erroneous copy of $Enc(\mathbf{d}_2)$. Use the decoder $Dec(\mathbf{m}') = \mathbf{d}_2$.

(5) Output $(d_1, \mathbf{d}_2)$.

The proof of correctness is similar to that in Sec. 6.3. The redundancy of the code is

$$\begin{aligned}
r(C) =& \log \binom{2^L}{M} - \log\left\lceil \frac{\prod_{i=1}^{M-1}(2^{L'} - iP)}{(M-1)!} \right\rceil \\
&- [M(L-L') - 4kL' - 8k\log(4kL') - o(\log(4kL')) \\
&- 4k\lceil \log ML \rceil - o(\log ML)] \\
\leq& 8k\log ML + (12k+2)\log M + O(k^3) + o(k\log ML).
\end{aligned}$$

Computing F_S^D

We now prove Lemma 6.5.1. The robust indexing algorithm for generating the indexing strings $\{x_{i,1}, \ldots, x_{i,L'}\}$ is the same as in Sec. 6.4 except that we replace the notations $N_H(\mathbf{a}, A)$ and Q, which are based on Hamming distance, with their deletion distance counterparts. For a string $\mathbf{c} \in \{0,1\}^\ell$ and a set of indices $\Delta = \{\delta_1, \ldots, \delta_r\} \subset [\ell]$, let $\mathbf{c}(\Delta)$ be the length $\ell - r$ subsequence of $\mathbf{c}$ obtained by deleting bits $(c_{\delta_1}, c_{\delta_2}, \ldots, c_{\delta_r})$ in $\mathbf{c}$.

For sequences $\mathbf{c}_1 \in \{0,1\}^{\ell_1}$ and $\mathbf{c}_2 \in \{0,1\}^{\ell_2}$ and nonnegative integers r_1, r_2, define the set

$$\begin{aligned}
\mathcal{I}(\mathbf{c}_1, \mathbf{c}_2, r_1, r_2) = \{(\Delta_1, \Delta_2) :& \Delta_1 \subseteq [\ell_1], |\Delta_1| \leq r_1, \Delta_2 \subseteq [\ell_2], |\Delta_2| \leq r_2, \\
& \mathbf{c}_1(\Delta_1) = \mathbf{c}_2(\Delta_2)\}
\end{aligned}$$

and the number

$$N(\mathbf{c}_1, \mathbf{c}_2, r_1, r_2) = |\mathcal{I}(\mathbf{c}_1, \mathbf{c}_2, r_1, r_2)|, \tag{6.14}$$

which is the number of ways to delete no more than r_1 and r_2 bits in $\mathbf{c}_1$ and $\mathbf{c}_2$, respectively, such that the resulting subsequences are the same. For a sequence $\mathbf{a} \in \{0,1\}^\ell$ of length $\ell \in [0, L']$ and a set of sequences $A \subset \{0,1\}^{L'}$, define

$$N_D(\mathbf{a}, A) = \sum_{\mathbf{c} \in A} \sum_{\mathbf{c}' : \mathbf{c}' \in \{0,1\}^{L'} \text{ and } (c'_1, \ldots, c_\ell) = \mathbf{a}} N(\mathbf{c}', \mathbf{c}, k, k).$$

For an empty sequence $\mathbf{a}$ and a sequence $\mathbf{c}$, we have that

$$N_D(\mathbf{a}, \mathbf{c}) = P \triangleq \sum_{r=0}^{k} \binom{L'}{r}^2 2^r, \tag{6.15}$$

since $N_D(\mathbf{a}, \mathbf{c})$ is the number of tuples $(\mathbf{c}', \Delta_1, \Delta_2)$ of sequences $\mathbf{c}' \in \{0,1\}^{L'}$ and index sets $\Delta_1, \Delta_2 \subset [L']$ such that after no more than k deletions in indices Δ_1 and Δ_2 in $\mathbf{c}$ and $\mathbf{c}'$, respectively, we obtain the same subsequence $\mathbf{c}(\Delta_1) = \mathbf{c}'(\Delta_2)$.

The algorithm for computing F_S^D is the same as that for computing F_S^H, by replacing the numbers $N_H((a_{i,1}, \ldots, a_{i,\ell-1}, 0), \{\mathbf{a}_j\}_{j=1}^{i-1})$ and Q with numbers $N_D((a_{i,1}, \ldots, a_{i,\ell-1}, 0), \{\mathbf{a}_j\}_{j=1}^{i-1})$ and P. To prove the correctness of the algorithm, we need to show that $N_D(\mathbf{a}, A)$ satisfies the two properties similar to the ones in Eq. (6.5) and Eq. (6.6). The first is that

$$N_D(\mathbf{a}, A) = N_D((\mathbf{a}, 0), A) + N_D((\mathbf{a}, 1), A) \tag{6.16}$$

for a sequence $\mathbf{a} \in \{0,1\}^\ell$ of length $\ell \in [L'-1]$ and a set $A \subset \{0,1\}^{L'}$, which is a deletion counterpart of Eq. (6.5). This can be proved by noticing that

$$N_D(\mathbf{a}, A) = \sum_{\mathbf{c}' : \mathbf{c}' \in \{0,1\}^{L'} \text{ and } (c'_1, \ldots, c'_\ell) = \mathbf{a}} \sum_{\mathbf{c} \in A} N(\mathbf{c}', \mathbf{c}, k, k)$$

and that for every sequence $\mathbf{c}' \in \{0,1\}^{L'}$ that satisfies $(c'_1, \ldots, c'_\ell) = \mathbf{a}$, we have either $c'_{\ell+1} = 1$ or $c'_{\ell+1} = 0$.

The second property is that the number $N_D(\mathbf{a}, A)$ is computable in polynomial time. Since obtaining an explicit expression as in Eq. (6.6) is challenging, we compute the number $N_D(\mathbf{a}, \mathbf{c})$ using dynamic programming for two sequences $\mathbf{a} \in \{0,1\}^\ell$ and $\mathbf{c} \in \{0,1\}^{L'}$ such that $\ell \in [0, L']$. Given $\mathbf{a}$ and $\mathbf{c}$, we compute

$$\begin{aligned} &n(k_1, k_2, r_1, r_2) \\ &= \sum_{\mathbf{c}' : \mathbf{c}' \in \{0,1\}^{L'-\ell+k_1} \text{ and } (c'_1, \ldots, c'_{k_1}) = (a_{\ell-k_1+1}, \ldots, a_\ell)} N(\mathbf{c}', (c_{L'-k_2+1}, \ldots, c_{L'}), r_1, r_2). \end{aligned}$$

Note that $N_D(\mathbf{a}, \mathbf{c}) = n(\ell, L', k, k)$. In addition, by definition of $N_D(\mathbf{a}, A)$, we have that $N_D(\mathbf{a}, A) = \sum_{\mathbf{c} \in A} N_D(\mathbf{a}, \mathbf{c})$. Hence, $N_D(\mathbf{a}, A)$ can be computed efficiently when $N_D(\mathbf{a}, \mathbf{c})$ is computed.

For $k_1 = 0$, we have that

$$n(0, k_2, r_1, r_2) = \sum_{\mathbf{c}' : \mathbf{c}' \in \{0,1\}^{L'-\ell}} N(\mathbf{c}', (c_{L'-k_2+1}, \ldots, c_{L'}), r_1, r_2), \tag{6.17}$$

which by Eq. (6.14) equals 0 when $L' - \ell - r_1 > k_2$ or $k_2 - r_2 > L' - \ell$. When $L' - \ell - r_1 \leq k_2$ and $k_2 - r_2 \leq L' - \ell$, we show that

$$n(0, k_2, r_1, r_2) = \sum_{i=k_2-(L'-\ell)}^{r_2} \binom{k_2}{i} \binom{L' - \ell}{L' - \ell - (k_2 - i)} 2^{L'-\ell-(k_2-i)}, \tag{6.18}$$

for $k_2 \geq L' - \ell$ and that

$$n(0, k_2, r_1, r_2) = \sum_{i=L'-\ell-k_2}^{r_1} \binom{k_2}{k_2 - (L' - \ell - i)} \binom{L' - \ell}{i} 2^i, \tag{6.19}$$

for $k_2 < L' - \ell$. For $k_2 \geq L' - \ell$ and sets $(\Delta_1, \Delta_2) \in \mathcal{I}(\mathbf{c}', \mathbf{c} = (c_{L'-k_2+1}, \ldots, c_{L'}), r_1, r_2)$, the cardinality $|\Delta_2|$ satisfies $k_2 - (L' - \ell) \leq |\Delta_2| \leq r_2$ because

$$\mathbf{c}'(\Delta_1) = (c_{L'-k_2+1}, \ldots, c_{L'})(\Delta_2).$$

For given $|\Delta_2|$, there are $\binom{k_2}{|\Delta_2|}$ ways to select Δ_2 and $\binom{L'-\ell}{L'-\ell-(k_2-\Delta_2)}$ choices of Δ_1. Moreover, given $\mathbf{c}$, Δ_1, and Δ_2, there are $2^{L'-\ell-(k_2-\Delta_2)}$ choices of $\mathbf{c}'$ such that $\mathbf{c}(\Delta_2) = \mathbf{c}'(\Delta_1)$. Hence we have Eq. (6.18). Similarly, we have Eq. (6.19). Therefore, the number $n_(k_1, k_2, r_1, r_2)$ can be computed when $k_1 = 0$.

For $k_1 > 0$, we compute n_{k_1,k_2,r_1,r_2} iteratively from $k_1 = 0$ to $k_1 = \ell$ using the following recursion.

$$\begin{aligned} n(k_1, k_2, r_1, r_2) = &\sum_{k:k\in[L'-k_2+1,L'],c_k=a_{\ell-k_1+1}} n(k_1 - 1, L' - k, r_1, r_2 - k + L' - k_2 + 1) \\ &+ 2n(k_1 - 1, k_2, r_1 - 1, r_2), \end{aligned} \tag{6.20}$$

where $n(k', k'', r', r'') = 0$ if $r' < 0$ or $r'' < 0$. Note that for any $(\Delta_1, \Delta_2) \in \mathcal{I}(\mathbf{c}', \mathbf{c} = (c_{L'-k_2+1}, \ldots, c_{L'}), r_1, r_2)$, we have either $1 \in \Delta_1$ or $1 \notin \Delta_1$. When $1 \in \Delta_1$, then $\mathbf{c}''(\Delta_1 \backslash \{1\} - 1) = (c_{L'-k_2+1}, \ldots, c_{L'})(\Delta_2)$, where $\mathbf{c}'' = (c'_2, \ldots, c'_{L'-\ell+k_1})$ and $\Delta - i = \{j - i : j \in \Delta\}$ for any set Δ and integer i. Note that there are $n(k_1-1, k_2, r_1-1, r_2)$ choices of $(\mathbf{c}'', \Delta_1\backslash\{1\} - 1, \Delta_2)$ such that $(c''_1, \ldots, c''_{k_1-1}) = (a_{\ell-k_1+2,\ldots,a_\ell})$ and $\mathbf{c}''(\Delta_1\backslash\{1\}-1) = (c_{L'-k_2+1}, \ldots, c_{L'})(\Delta_2)$. Since c'_1 can be either 0 or 1 when $1 \in \Delta_1$. We have $2n(k_1 - 1, k_2, r_1 - 1, r_2)$ choices of $(\mathbf{c}', \Delta_1, \Delta_2)$ such that $(c'_2, \ldots, c'_{k_1}) = (c''_1, \ldots, c''_{k_1-1}) = (a_{l-k_1+2,\ldots,a_\ell})$ and $\mathbf{c}'(\Delta_1) = (c_{L'-k_2+1}, \ldots, c_{L'})(\Delta_2)$, when $1 \in \Delta_1$. When $1 \notin \Delta_1$, Let k be the minimum index such that $k \in [L' - k_2 + 1, L']$ and $(k - L' + k_2) \notin \Delta_2$. Then, we have that $c_k = c'_1 = a_{l-k_1+1}$, $[1, k - L' + k_2 - 1] \in \Delta_2$, and $\mathbf{c}''(\Delta_1 - 1) = (c_{k+1}, \ldots, c_{L'})(\Delta_2\backslash[1, k - L' + k_2 - 1] - k + L' - k_2)$, where $\mathbf{c}'' = (c'_2, \ldots, c'_{L'-\ell+k_1})$. There are $n(k_1 - 1, L' - k, r_1, r_2 - k + L' - k_2 + 1)$ choices of $(\mathbf{c}'', \Delta_1 - 1, \Delta_2\backslash[1, k - L' + k_2 - 1] - k + L' - k_2)$ such that $\mathbf{c}''(\Delta_1 - 1) =$

$(c_{k+1},\dots,c_{L'})(\Delta_2\backslash[1,k-L'+k_2-1]-k+L'-k_2)$ and that $(c''_1,\dots,c''_{k_1-1}) = (a_{\ell-k_1+2,\dots,a_\ell})$. Therefore, there are $n(k_1-1,L'-k,r_1,r_2-k+L'-k_2+1)$ choices of $(\mathbf{c}',\Delta_1,\Delta_2)$ such that $(c'_1,\dots,c'_{k_1}) = (a_{\ell-k_1+1},\dots,a_\ell)$ and $\mathbf{c}'(\Delta_1) = (c_{L'-k_2+1},\dots,c_{L'})(\Delta_2)$. Note that for each k satisfying $k \in [L'-k_2+1,L']$ and $c_k = c'_1 = a_{\ell-k_1+1}$, there are $n(k_1-1,L'-k,r_1,r_2-k+L'-k_2+1)$ choices of such $(\mathbf{c}',\Delta_1,\Delta_2)$. In addition, different k corresponds to different choices since k is the minimum index such that $(k-L'+k_2)\notin\Delta_2$. Hence, we have (6.4).

By Eq. (6.17), (6.18), (6.19), and (6.4), the number $N(\mathbf{a},\mathbf{c}) = n(\ell,L',k,k)$ can be recursively computed for any $\mathbf{a}\in\{0,1\}^\ell$ and $\mathbf{c}\in\{0,1\}^{L'}$. Therefore, the encoding/decoding can be computed in $poly(M,L')$ time.

We are now ready to present the algorithm that computes $F_S^D(d)$ for an integer $d \in \left[\binom{2^{L'}-(M-1)P+M-1}{M-1}\right]$. The algorithm is the same as the encoding procedure in Sec. 6.4, by replacing $N_H(\mathbf{a},A)$ with $N_D(\mathbf{a},A)$ for any sequence $\mathbf{a}$ and set of sequences A. In addition, the integers q_i are generated such that $q_1 = 2^{L'}$ and $q_{i+1}-q_i > P$ for $i\in[M-1]$. Such q_i, $i\in[M]$ can be generated following the same argument in Lemma 6.4.1, since $d \in \left[\binom{2^{L'}-(M-1)P+M-1}{M-1}\right]$. Given integers q_i, $i\in[M]$, satisfying $q_1 = 2^{L'}$ and $q_{i+1}-q_i > P$ for $i\in[M-1]$, the encoding procedure for generating $\{\mathbf{a}_1,\dots,\mathbf{a}_M\}$ is given as follows.

Encoding:

```
for $i \in [M]$, do
$q = q_i$.
   for $\ell \in [L']$, do
     if $2^{L'-\ell} - N_D((a_{i,1},\dots,a_{i,\ell-1},0),\{\mathbf{a}_j\}_{j=1}^{i-1}) \geq q$,
       then $a_{i,\ell} = 0$.
     else
       $q = q - (2^{L'-\ell} - N_D((a_{i,1},\dots,a_{i,\ell-1},0),\{\mathbf{a}_j\}_{j=1}^{i-1}))$,
       $a_{i,\ell} = 1$.
     end if
   end for
end for
return $\{\mathbf{a}_1, \ldots, \mathbf{a}_M\}$.
```

The correctness of the encoding procedure follows similar argument to the one in Sec. 6.4. We prove that the input $(q_1, \ldots, q_M)$ and output $\{\mathbf{a}_1, \ldots, \mathbf{a}_M\}$ satisfy

$$\text{decimal}(\mathbf{a}_i) = q_i - 1 + \sum_{\ell : a_{i,\ell}=1 \text{ and } \ell \in [L']} N_D((a_{i,1}, \ldots, a_{i,\ell-1}, 0), \{\mathbf{a}_j\}_{j=1}^{i-1}) \quad (6.21)$$

and $\{\mathbf{a}_1, \ldots, \mathbf{a}_M\} \in \mathcal{S}^D$. The following is a deletion metric version of 6.4.3, by replacing $N_H(\mathbf{a}, A)$ with $N_D(\mathbf{a}, A)$ for any sequence $\mathbf{a} \in \{0,1\}^\ell$ and set $A \in \{0,1\}^{L'}$.

Lemma 6.5.3. *After the ℓ-th, $\ell \in [L']$, inner for loop in the i-th, $i \in [M]$, outer for loop in the encoding procedure, we have that*

$$0 < q \leq 2^{L'-\ell} - N_D((a_{i,1}, \ldots, a_{i,\ell}), \{\mathbf{a}_j\}_{j=1}^{i-1}). \quad (6.22)$$

At the end of the i-th outer for loop, we have that $q = 1$.

Proof. The proof is the same as that of Lemma 6.4.3, by noticing that

$$\begin{aligned} N_D(0, \{\mathbf{a}_j\}_{j=1}^{i-1}) + N_D(1, \{\mathbf{a}_j\}_{j=1}^{i-1}) &= \sum_{j=1}^{i-1} N_D(, \mathbf{a}_j) \\ &\overset{(a)}{=} (i-1)P, \end{aligned}$$

where is the empty sequence and (a) follows from (6.15) and the fact that $N_D(\mathbf{a}, A) = \sum_{\mathbf{c} \in A} N_D(\mathbf{a}, \mathbf{c})$. In addition, we have (6.16), which is the deletion metric version of (6.5). The rest of the proof follows the same as in 6.4.3. □

From Lemma 6.5.3, we have

$$q = 2^{L'-L'} - N_D(\mathbf{a}_i, \{\mathbf{a}_j\}_{j=1}^{i-1}) = 1,$$

at the end of the i-th outer for-loop, $i \in [M]$. Hence, $N_D(\mathbf{a}_i, \{\mathbf{a}_j\}_{j=1}^{i-1}) = 0$ for $i \in [M]$ and $\mathcal{D}_k(\mathbf{a}_i) \cap \mathcal{D}_k(\mathbf{a}_j) = \emptyset$ for any $i \neq j$, $i, j \in [M]$. Then, we have that $\{\mathbf{a}_i\}_{i=1}^M \in \mathcal{S}^D$. In addition, similar to Lemma 6.4.4, we can use Lemma 6.5.3 to show that the output $\{\mathbf{a}_i\}_{i=1}^M$ satisfies Eq. (6.21).

Therefore, we have the following decoding algorithm, similar to the one in Sec. 6.4.

Decoding:

(1) Order the strings $\{\mathbf{a}_i\}_{i=1}^{M}$ such that $\mathbf{a}_1 > \mathbf{a}_2 > \ldots > \mathbf{a}_M$.

(2) For $i \in [M]$,

$$q_i = \text{decimal}(\mathbf{a}_i) + 1 + \sum_{\ell: a_{i,\ell}=1 \text{ and } \ell \in [L']} N_D((a_{i,1}, \ldots, a_{i,\ell-1}, 0), \{\mathbf{a}_j\}_{j=1}^{i-1}). \tag{6.23}$$

Finally, the correctness of decoding is guaranteed by (6.21) and the fact that $\mathbf{a}_1 > \mathbf{a}_2 > \ldots > \mathbf{a}_M$, where $\mathbf{a}_i$ is the output generated in the i-th outer-loop. The latter follows similar proof to the one in Sec. 6.4. Suppose there exists $i_1 > i_2$ such that $\mathbf{a}_{i_1} > \mathbf{a}_{i_2}$ alphabetically. Then we have that

$$\begin{aligned} q_{i_2} - q_{i_1} < & \sum_{\ell: a_{i_1,\ell}=1 \text{ and } \ell \in [\ell^*]} (2^{L'-\ell} \\ & - N_D((a_{i_1,1}, \ldots, a_{i_1,\ell-1}, 0), \{\mathbf{a}_j\}_{j=1}^{i_2-1})) \\ & - \sum_{\ell: a_{i_1,\ell}=1 \text{ and } \ell \in [\ell^*]} (2^{L'-\ell} - N_D((a_{i_1,1}, \ldots, a_{i_1,\ell-1}, 0), \{\mathbf{a}_j\}_{j=1}^{i_1-1})) \\ = & \sum_{\ell: a_{i_1,\ell}=1 \text{ and } \ell \in [\ell^*]} (N_D((a_{i_1,1}, \ldots, a_{i_1,\ell-1}, 0), \{\mathbf{a}_j\}_{j=1}^{i_1-1}) \\ & - N_D((a_{i_1,1}, \ldots, a_{i_1,\ell-1}, 0), \{\mathbf{a}_j\}_{j=1}^{i_2-1})) \\ = & \sum_{\ell: a_{i_1,\ell}=1 \text{ and } \ell \in [\ell^*]} N_D((a_{i_1,1}, \ldots, a_{i_1,\ell-1}, 0), \{\mathbf{a}_j\}_{j=i_2}^{i_1-1}) \\ \overset{(a)}{\leq} & N_D(\emptyset, \{\mathbf{a}_j\}_{j=i_2}^{i_1-1}) \\ \overset{(b)}{\leq} & (i_1 - i_2)P, \end{aligned} \tag{6.24}$$

where $\emptyset$ is the empty sequence and (a) follows from the definition of $N_D(\mathbf{a}, A)$ and the fact that the strings which have $(a_{i_1,1}, \ldots, a_{i_1,\ell_1-1}, 0)$ and $(a_{i_1,1}, \ldots, a_{i_1,\ell_2-1}, 0)$ as prefixes, respectively, are different. Inequality (b) follows from (6.15) and the fact that $N_D(\mathbf{a}, A) = \sum_{\mathbf{c} \in A} N_D(\mathbf{a}, \mathbf{c})$.

6.6 Conclusion

In this chapter, we provided order-wise optimal codes correcting k substitutions, and a combination of at most k deletion or insertions, respectively, over an unordered set of M sequences, each of length L. There are limitations on the parameters (k, M, L), where our results apply. It is interesting and desirable to see if such optimality holds for broader set of parameters (k, M, L).

6.7 Appendix

Proof of Eq. (6.2)

$$\begin{aligned}
r(C) =& \log\binom{2^L}{M} - \log\lceil\frac{\prod_{i=1}^{M-1}(2^{L'}-iQ)}{(M-1)!}\rceil \\
&- [M(L-L') - 4kL' - 2k\lceil\log ML\rceil] \\
\le& \log\frac{2^{LM}}{M!} - \log\frac{(2^{L'}-MQ)^{M-1}}{(M-1)!} \\
&- [M(L-L') - 4kL' - 2k(\log ML + 1)] \\
=& ML' - \log(2^{L'}-MQ)^{M-1} + 4kL' \\
&+ 2k\log ML + 2k - \log M \\
=& \log\frac{2^{L'(M-1)}}{(2^{L'}-MQ)^{M-1}} + L' + 4kL' \\
&+ 2k\log ML + 2k - \log M \\
=& \frac{(M-1)MQ}{2^{L'}-MQ}\log(1+\frac{MQ}{2^{L'}-MQ})^{\frac{2^{L'}-MQ}{MQ}} + L' + 4kL' \\
&+ 2k\log ML + 2k - \log M \\
\overset{(a)}{\le}& \frac{(M-1)}{M}\log(1+\frac{1}{M})^M + L' + 4kL' \\
&+ 2k\log ML + 2k - \log M \\
\le& \log e + L' + 4kL' + 2k\log ML + 1 + 2k - \log M \\
=& 2k\log ML + (12k+2)\log M + O(k^3) + O(k\log\log ML),
\end{aligned}$$

where (a) follows from the following inequality

$$M^2(3\log M + 4k^2 + 1)^{2k} \le 2^{3\log M + 4k^2 + 1} \tag{6.25}$$

and the fact that the function $f(x) = (1+x)^{1/x}$ is decreasing in x for $0 < x$. Eq. (6.25) is proved as follows.

Rewrite Eq. (6.25) as

$$(3\log M + 4k^2 + 1)^{2k} \le 2^{\log M + 4k^2 + 1}. \tag{6.26}$$

Define functions $g(y,k) = \ln(3y + 4k^2 + 1)^{2k}$ and $h(y,k) = \ln 2^{y+4k^2+1}$. Then we have that

$$\partial h(y,k)/\partial y - \partial g(y,k)/\partial y = \ln 2 - 6k/(3y + 4k^2 + 1),$$

which is positive for $y \geq 1$ and $k \geq 2$. Therefore, for $k \geq 2$ and $y \geq 1$, we have that

$$h(y,k) - g(y,k) \geq h(1,k) - g(1,k).$$

Furthermore,

$$\begin{aligned}\partial h(1,k)/\partial k - \partial g(1,k)/\partial k =&(8\ln 2)k - 2\ln(4k^2+4) - 16k^2/(4k^2+4)\\ >&(8\ln 2)k - 2\ln(5k^2) - 4\\ =&4(k-1-\ln k) + (8\ln 2 - 4)k - 2\ln 5\\ \overset{(a)}{\geq}&(8\ln 2 - 4)k - 2\ln 5,\end{aligned}$$

where (a) follows since $k = e^{\ln k} \geq 1 + \ln k$. Since $(8\ln 2 - 4)k - 2\ln 5$ is positive for $k \geq 3$, we have that $h(1,k)/\partial k > \partial g(1,k)/\partial k$ for $k \geq 3$. It then follows that $h(1,k) - g(1,k) \geq \min\{h(1,2) - g(1,2), h(1,3) - g(1,3)\} > 0$ for $k \geq 2$. Hence $h(y,k) > g(y,k)$ for $y \geq 1$ and $k \geq 2$, which implies that Eq. (6.26) holds when $M \geq 2$ and $k \geq 2$.

Next we show that Eq. (6.26) holds when $M = 1$ or $k = 1$. When $M = 1$, we have that $\log M = 0$ and that

$$\begin{aligned}\partial h(0,k)/\partial k - \partial g(0,k)/\partial k =&(8\ln 2)k - 2\ln(4k^2+1) - 16k^2/(4k^2+1)\\ >&(8\ln 2)k - 2\ln(5k^2) - 4\\ =&4(k-1-\ln k) + (8\ln 2 - 4)k - 2\ln 5\\ \geq&(8\ln 2 - 4)k - 2\ln 5,\end{aligned}$$

which is positive when $k \geq 3$. Therefore, we have that $h(0,k) - g(0,k) \geq \min\{h(0,1) - g(0,1), h(0,2) - g(0,2), h(0,3) - g(0,3)\} > 0$. Hence Eq.(6.26) holds when $M = 1$.

When $k = 1$ we have that

$$\begin{aligned}2^{\log M + 4k^2 + 1} =&32(1 + + \sum_{i=1}^{\infty} \log^i M/i!)\\ \geq&32(1 + \log M + \log^2 M/2)\\ \geq&(3\log M + 5)^2\\ =&(3\log M + 4k^2 + 1)^{2k}.\end{aligned}$$

Hence, Eq. (6.26) and Eq. (6.25) holds. We now finish the proof of Eq. (6.2).

Chapter 7

TRACE RECONSTRUCTION

7.1 Introduction

The trace reconstruction problem seeks to recover an unknown string $\mathbf{x} \in \{0,1\}^n$, given multiple independent noisy samples or traces of $\mathbf{x}$. In this chapter, a noisy sample is obtained by passing $\mathbf{x}$ through a deletion channel, which randomly and independently deletes each bit of $\mathbf{x}$ with probability q. We are interested in how many samples are needed to recover $\mathbf{x}$ with high probability.

The trace reconstruction problem was introduced in [5] and proposed earlier in [66] under an adversarial setting. It has been receiving increased attention recently due to its application in DNA sequencing [8] and DNA storage under nanopore sequencing [73, 99]. Also, there are many significant results on trace reconstruction and its variants and generalizations, such as coding for trace reconstruction [23] and population recovery [4]. For average case trace reconstruction, where the reconstruction error probability is averaged over all choices of $\mathbf{x} \in \{0,1\}^n$, the state of the art upper and lower bounds on the number of samples are $\exp(O(\log^{\frac{1}{3}}(n)))$ [48] and $\Omega(\frac{\log^{\frac{5}{2}}(n)}{(\log\log n)^7})$ [16] respectively.

Despite the progress for average cases, the trace reconstruction problem proved to be highly nontrivial in worst cases, where the reconstruction error probability goes to zero for arbitrary choice of $\mathbf{x}$. For small deletion probabilities, the work in [20] showed that polynomial number of samples suffice when $q \leq n^{-(\frac{1}{3}+\epsilon)}$ for some $\epsilon > 0$, improving the result in [5] for $q \leq n^{-(\frac{1}{2}+\epsilon)}$ and some $\epsilon > 0$. When the deletion probability becomes constant, there is still an exponential gap between the upper and lower bounds on the number of samples needed. The first achievable sample size for constant deletion probability q is $\exp(\tilde{O}(n^{\frac{1}{2}}))$ [69], which was improved to $\exp(\Theta(n^{\frac{1}{3}}))$ in independent and simultaneous works [27] and [70]. Both [27] and [70] studied mean-based algorithms, which use single-bit statistics in traces, for reconstruction. They showed that $\exp(O(n^{\frac{1}{3}}))$ is the best sample size achieved by mean-based algorithms. A novel approach in [27] and [70] is to relate single-bit statistics to complex polynomial analysis, and borrow results from [10] on complex analysis. This approach was further developed in [15], where multi-bit statistics were considered. The current best upper bound on the sample size is $\exp(\tilde{O}(n^{\frac{1}{5}}))$

[15], while the best lower bound $\Omega(\frac{n^{\frac{3}{2}}}{\log^7 n})$ [16] is orders of magnitude away from the upper bound.

While the general trace reconstruction problem is hard to solve, in this chapter, we focus on a variant of the trace reconstruction problem with an edit distance constraint. Specifically, the goal is to recover the string $\mathbf{x}$ by using its noisy samples and additional information of a given string $\mathbf{y}$, which is known to be within a bounded distance from $\mathbf{x}$. The edit distance between two strings is commonly defined as the minimum number of deletions, insertions, or substitutions that transform one string into another. In this chapter, we consider only deletion/insertion for convenience, as a substitution is an insertion followed by a deletion. We say that a string $\mathbf{x}$ is within edit distance k to a string $\mathbf{y}$, denoted as $\mathbf{x} \in \mathcal{B}_k(\mathbf{y})$, if $\mathbf{x}$ can be obtained from $\mathbf{y}$ after at most k deletions and k insertions. Note that the general trace reconstruction problem considers cases where $k = n$.

The setting considered in this chapter arises in many practical scenarios in genome sequencing, where one needs to recover an individual genome sequence of a species, given a reference genome sequence that represents the species [2]. Normally, the genome sequences of a species share some similarity and most of them can be considered to be within a bounded edit distance from the reference genome. One example is the Human Genome Project, where a human reference genome is provided to study the difference between individual genomes. Complementary to the problem we consider, the work in [26] studied approximate trace reconstruction, which aims to find an estimate within a given edit distance to the true string. Note that such an estimate, together with an algorithm to distinguish two strings within edit distance k, establishes a solution to the general trace reconstruction problem.

As indicated in [27, 37, 49, 57, 70], the problem of worst-case trace reconstruction is essentially equivalent to a hypothesis testing problem of distinguishing any two strings using noisy samples. More specifically, the sample complexity needed for trace reconstruction is at most $poly(n)$ times the sample complexity needed to distinguish arbitrary two strings. The same equivalence holds in our setting as well, where a reference string $\mathbf{y}$ is known and close to $\mathbf{x}$ in edit distance. Hence, for convenience, we consider the problem in the form of distinguishing any two strings $\mathbf{x} \in \{0, 1\}^n$ and $\mathbf{y} \in \{0, 1\}^n$ when $\mathbf{x}$ is within edit distance k to $\mathbf{y}$. One special case of the problem is to distinguish two strings within Hamming distance k, which was addressed in [57] and $n^{O(k)}$ sample complexity was achieved. Recently, an independent work [37] studied the limitations of mean-based algorithms (see [27] and [70]) in distinguishing two strings with bounded edit distance. It was shown that mean-based algorithms need at least $n^{O(\log n)}$ traces to distinguish two strings with edit distance of even 4. The paper [37] also showed that $n^{O(k^2)}$ suffices to distinguish two strings $\mathbf{x} \in \{0,1\}^n$ and $\mathbf{y} \in \{0,1\}^n$ with special block structures, if $\mathbf{x} \in \mathcal{B}_k(\mathbf{y})$. Yet, as pointed out in [26], it is an open problem whether $n^{O(k)}$ samples suffice to recover a string that is within edit distance k to a known string.

The main contribution of this chapter is an affirmative answer to this question. We show that distinguishing two sequences within edit distance k needs at most $n^{O(k)}$ samples. The result is stated by the following theorem.

Theorem 7.1.1. *Let $\mathbf{x} \in \{0,1\}^n$ and $\mathbf{y} \in \{0,1\}^n$ be two strings satisfying $\mathbf{x} \in \mathcal{B}_k(\mathbf{y})$. Then strings $\mathbf{x}$ and $\mathbf{y}$ can be distinguished with high probability, given $n^{O(k)}$ independent noisy samples, each obtained by passing $\mathbf{x}$ through a deletion channel with deletion probability $q < 1$.*

Remark 7.1.1. *Theorem 7.1.1 holds for any string $\mathbf{y}$ that can be obtained from $\mathbf{x}$ after at most k deletions or insertions. The length of $\mathbf{y}$ is not necessarily n. Yet by definition of the trace reconstruction problem, we focus on length n strings $\mathbf{x}$ and $\mathbf{y}$.*

The approach we take follows a similar method to that in [15, 27, 37, 70], in the sense that we derive bounds on multi-bit statistics through complex analysis of a special class of polynomials. Yet, the complex analysis in this chapter differs from those in [15, 27, 37, 70] in the following two ways. Firstly, we make use of the fact that the polynomial is related to a number theoretic problem called the Prouhet-Tarry-Escott problem [9], which is also noted in [37]. This allows us to link the problem to our previous result on deletion codes [86], where we showed that two constrained strings can be distinguished using weighted sums of powers, which is similar in form to the Prouhet-Tarry-Escott problem. Secondly, to find the maximum value of the polynomial, we let the complex variable take values on a small circle around the point 1, while the work in [15, 27, 37, 70] analyzes the complex polynomial on a unit circle. By doing this, we are able to improve the $n^{O(k^2)}$ bound in [37] to $n^{O(k)}$.

The rest of the chapter is organized as follows. In Sec. 7.2 we provide an introduction to the techniques and the lemmas needed to prove Theorem 7.1.1. In Sec. 7.3, the proof of Theorem 7.1.1 is given. Sec. 7.4 presents the proof of a critical lemma on complex analysis. Sec. 7.5 concludes the chapter.

7.2 Proof Techniques and Lemmas

In this section we present a brief introduction to the techniques and key lemmas needed in proving Theorem 1. For strings $\mathbf{x} \in \{0,1\}^n$ and $\mathbf{y} \in \{0,1\}^n$, let $\tilde{X} = (\tilde{X}_1, \ldots, \tilde{X}_n)$ and $\tilde{Y} = (\tilde{Y}_1, \ldots, \tilde{Y}_n)$ denote the sample obtained by passing $\mathbf{x}$ and $\mathbf{y}$ through the deletion channel respectively. We have $\tilde{X}_i = \emptyset$ or $\tilde{Y}_j = \emptyset$ if i or j is larger than the length of $\tilde{X}$ or $\tilde{Y}$, respectively. Note that $\tilde{X}$ and $\tilde{Y}$ are sequences of random variables that describe the probability distributions of the samples.

The techniques we use were originated in [27, 70], which presented the following identity

$$\begin{aligned}\mathbb{E}_{\tilde{X}}\Big[\sum_{i=1}^{n} \tilde{X}_i (\frac{z-q}{1-q})^i\Big] &= (1-q)\sum_{i=1}^{n} x_i z^i \\ &\triangleq f^s_{\mathbf{x}}(z), \end{aligned} \tag{7.1}$$

for a sequence $\mathbf{x}$ and a complex number z. The identity (7.1) links the analysis of single bit statistics $\{E_{\tilde{X}}[\tilde{X}_i]\}_{i=1}^n$ to that of complex polynomials. As a result, a lower bound on the maximal difference between single bit statistics $\max_{1\le i\le n} |E_{\tilde{X}}[\tilde{X}_i] - E_{\tilde{Y}}[\tilde{Y}_i]|$ can be obtained through analyzing the maximal value of the polynomial $f^s_{\mathbf{x}}(z) - f^s_{\mathbf{y}}(z)$ on a unit disk, a problem referred to as Littlewood type problems and studied in [10, 11]. Generalizing the approach in [27, 70], the papers [15] and [21] presented multi-bit statistics counterparts of (7.1). In this chapter, we consider the version from [15], stated in the following lemma.

Lemma 7.2.1. *[15] For integer $\ell \ge 1$, complex numbers $z_1, \ldots, z_\ell$, and sequences $\mathbf{x} \in \{0,1\}^n$ and $\mathbf{w} \in \{0,1\}^\ell$, we have*

$$\begin{aligned}&\mathbb{E}_{\tilde{X}}\Big[(1-q)^{-\ell} \sum_{1\le i_1<\ldots<i_\ell\le n} \mathbb{1}_{\tilde{X}_{i_j}=w_j, \forall j\in[\ell]} \\ &\qquad\cdot (\frac{z_1-q}{1-q})^{i_1} \prod_{j=2}^{\ell} (\frac{z_j-q}{1-q})^{i_j-i_{j-1}-1}\Big] \\ &= \sum_{1\le j_1<\ldots<j_\ell\le n} \mathbb{1}_{x_{j_h}=w_h, \forall h\in[\ell]} z_1^{j_1} \prod_{h=1}^{\ell} z_j^{j_h-j_{h-1}-1} \\ &\triangleq f_{\mathbf{x},\mathbf{w}}(z_1, \ldots, z_\ell), \end{aligned} \tag{7.2}$$

where $[\ell] = \{1, \ldots, \ell\}$ and $i : i+\ell-1 = \{i, \ldots, i+\ell-1\}$. For any statement E, the variable $\mathbb{1}_E = 1$ iff E holds true.

By taking $z_2 = \ldots = z_\ell = 0$ in (7.2), we obtain

$$f_{\mathbf{x},\mathbf{w}}(z,0,\ldots,0) = \sum_{i=1}^{n-\ell+1} \mathbb{1}_{\mathbf{x}_{i:i+\ell-1}=\mathbf{w}} z^i. \tag{7.3}$$

Similar to the arguments in [15], we prove Theorem 7.1.1 by analyzing the polynomial $f_{\mathbf{x},\mathbf{w}}(z,0,\ldots,0) - f_{\mathbf{y},\mathbf{w}}(z,0,\ldots,0)$ associated with the multi-bit statistics in (7.3). Note that the polynomial $f_{\mathbf{x},\mathbf{w}}(z,0,\ldots,0) - f_{\mathbf{y},\mathbf{w}}(z,0,\ldots,0)$ is single variate. The way in which the polynomial is analyzed in this chapter deviates from that in [15]. While the paper [15] taylored the complex analysis arguments in [11] to obtain improved bounds, in this chapter, we exploit number theoretic properties of two strings $\mathbf{x}$ and $\mathbf{y}$ within edit distance k.

In Ch. 3, we showed implicitly that the weighted sums of powers $\sum_{i=1}^n i^j x_i$, $j \in \{0,\ldots,O(k)\}$ can be used to distinguish two constrained strings $\mathbf{x}$ and $\mathbf{y}$ within edit distance k. The following lemma makes this statement explicit. Let $\mathcal{R}_{n,k}$ denote the set of length n strings such that any two 1 entries in each string are separated by a 0 run of length at least $k-1$.

Lemma 7.2.2. *For distinct strings $\mathbf{x},\mathbf{y} \in \mathcal{R}_{n,6k}$, if $\mathbf{x} \in \mathcal{B}_{6k}(\mathbf{y})$, then there exists an integer $m \in [12k+1]$ such that $\sum_{i=1}^n i^m x_i \neq \sum_{i=1}^n i^m y_i$.*

Proof. Suppose on the contrary, we have that $\sum_{i=1}^n i^m x_i = \sum_{i=1}^n i^m y_i$ for all $m \in [12k+1]$. Then, we have that

$$\sum_{i=1}^n \Big(\sum_{j=1}^i j^{m'}\Big) x_i = \sum_{i=1}^n \Big(\sum_{j=1}^i j^{m'}\Big) y_i \tag{7.4}$$

for all $m' \in \{0,\ldots,12k\}$. This is because $\sum_{j=1}^i j^{m'}$ is a weighted sum of $i^1,\ldots,i^{m'+1}$ for any $m' \in \{0,\ldots,12k+1\}$ (Faulhaber's formula). Next, we borrow a result from [86].

Proposition 7.2.1. *[86] For sequences $\mathbf{x},\mathbf{y} \in \mathcal{R}_{n,3k}$, if $\mathbf{y} \in \mathcal{B}_{3k}(\mathbf{x})$ and $\sum_{i=1}^n (\sum_{j=1}^i j^m) x_i = \sum_{i=1}^n (\sum_{j=1}^i j^m) y_i$ for $m \in \{0,\ldots,6k\}$, then $\mathbf{x} = \mathbf{y}$.*

Note that $\mathcal{B}_{5k}(\mathbf{x}) \subseteq \mathcal{B}_{6k}(\mathbf{x})$. Since (7.4) holds, we apply Proposition 7.2.1 with $k = 2k$ and conclude that $\mathbf{x} = \mathbf{y}$, which contradicts the fact that $\mathbf{x}$ and $\mathbf{y}$ are distinct. □

Interestingly, the following result from [9] connects the sums of powers of two sets of integers that appear in Lemma 7.2.2 to the number of roots of a polynomial at 1.

It allows us to combine the number theoretic result with further complex analysis, which will be given in Lemma 7.2.6. The lemma can be proved by checking the i-th, $i \in [m]$, derivative of the polynomial $\sum_{i=1}^{s} z^{\alpha_i} - \sum_{i=1}^{t} z^{\beta_i}$ at point $z = 1$.

Lemma 7.2.3. *[9] Let $\{\alpha_1, \ldots, \alpha_s\}$ and $\{\beta_1, \ldots, \beta_s\}$ be two sets of integers. The following are equivalent:*

(a) $\sum_{i=1}^{s} \alpha_i^j = \sum_{i=1}^{s} \beta_i^j$ *for* $j \in [m-1]$.

(b) $(z-1)^m$ divides $\sum_{i=1}^{s} z^{\alpha_i} - \sum_{i=1}^{s} z^{\beta_i}$.

Remark 7.2.1. *The problem of finding two sets of integers $\{\alpha_1, \ldots, \alpha_s\}$ and $\{\beta_1, \ldots, \beta_s\}$ satisfying the statement (a) is called the Prouhet-Tarry-Escott problem [9]. This connection between the Prouhet-Tarry-Escott problem and the analysis of polynomials was also used in [37] and implicitly in [56].*

Lemma 7.2.2 requires that the strings $\mathbf{x}$ and $\mathbf{y}$ are within $\mathcal{R}(n, 6k)$, which does not hold in general. Following the same trick as in [15] and [86], we define an indicator vector as follows. For any sequences $\mathbf{x} \in \{0,1\}^n$ and $\mathbf{w} \in \{0,1\}^\ell$, define the length n vector

$$\mathbb{1}_{\mathbf{w}}(\mathbf{x})_i \triangleq \begin{cases} 1, & \text{if } \mathbf{x}_{i:i+\ell-1} = \mathbf{w}, \\ 0, & \text{else}, \end{cases}$$

for $i \in [n]$. Note that $\mathbb{1}_{\mathbf{w}}(\mathbf{x})_i = 0$ for $i \in \{n-\ell+2, \ldots, n\}$. It can be seen that the polynomial $f_{\mathbf{x},\mathbf{w}}(z, 0, \ldots, 0)$ related to multi-bit statistics is exactly the polynomial $f^s_{\mathbb{1}_{\mathbf{w}}(\mathbf{x})}(z)$ related to single-bit statistics. To apply Lemma 7.2.2, we need to find a $\mathbf{w}$ such that $\mathbb{1}_{\mathbf{w}}(\mathbf{x}) \in \mathcal{R}(n, 6k)$. The same as what the paper [15] did, we find such a $\mathbf{w}$ by using the following lemma from [78]. A string $\mathbf{w} \in \{0,1\}^\ell$ is said to have period a, if and only if $w_i = w_{i+a}$ for $i \in [\ell - a]$. Moreover, a string $\mathbf{w} \in \{0,1\}^\ell$ is said to be non-periodic, iff $\mathbf{w}$ does not have period a for $a \in [\lceil \frac{\ell}{2} \rceil - 1]$.

Lemma 7.2.4. *For any sequences $\mathbf{w} \in \{0,1\}^{2p-1}$, either $(\mathbf{w}, 0)$ or $(\mathbf{w}, 1)$ is non-periodic, where $(\mathbf{w}, 0)$ and $(\mathbf{w}, 1)$ is the string obtained by appending 0 and 1 to $\mathbf{w}$, respectively.*

Lemma 7.2.4 can be proved by definition of period. The claim that $\mathbb{1}_{\mathbf{w}}(\mathbf{x}) \in \mathcal{R}(n, p)$ follows from Lemma 7.2.4 and will be proved in Lemma 7.2.5. In addition, the edit distance between $\mathbb{1}_{\mathbf{w}}(\mathbf{x})$ and $\mathbb{1}_{\mathbf{w}}(\mathbf{y})$ needs to be bounded to apply Lemma 7.2.2. This is proved in the following lemma.

Lemma 7.2.5. *Let* $\mathbf{w} \in \{0,1\}^{2p}$ *be a non-periodic string. For two strings* $\mathbf{x}$ *and* $\mathbf{y} \in \mathcal{B}_k(\mathbf{x})$*, we have that*

(a) $\mathbb{1}_{\mathbf{w}}(\mathbf{x}) \in \mathcal{R}_{n,p}$.

(b) $\mathbb{1}_{\mathbf{w}}(\mathbf{y}) \in \mathcal{R}_{n,p}$.

(c) $\mathbb{1}_{\mathbf{w}}(\mathbf{x}) \in \mathcal{B}_{5k}(\mathbb{1}_{\mathbf{w}}(\mathbf{y}))$.

Proof. The statements (a) and (b) follow from the definition of vectors $\mathbb{1}_{\mathbf{w}}(\mathbf{x})$ and $\mathbb{1}_{\mathbf{w}}(\mathbf{y})$ and the fact that $\mathbf{w}$ is non-periodic. Suppose there are two 1 entries $\mathbb{1}_{\mathbf{w}}(\mathbf{x})_i$ and $\mathbb{1}_{\mathbf{w}}(\mathbf{x})_{i+a}$ in $\mathbb{1}_{\mathbf{w}}(\mathbf{x})$ that are separated by less than $p-1$ 0's, i.e., $a \leq p-1$. Then by definition of $\mathbb{1}_{\mathbf{w}}(\mathbf{x})$, we have that $\mathbf{x}_{i:i+2p-1} = \mathbf{w}$ and that $\mathbf{x}_{i+a:i+a+2p-1} = \mathbf{w}$. This implies that $w_j = x_{i+a+j-1} = w_{j+a}$ for $j \in [2p-a]$. Hence, the string $\mathbf{w}$ has period $a \leq p-1$, contradicting to the fact that $\mathbf{w}$ is non-periodic. Hence, we have that $\mathbb{1}_{\mathbf{w}}(\mathbf{x}) \in \mathcal{R}_{n,p}$, and similarly that $\mathbb{1}_{\mathbf{w}}(\mathbf{y}) \in \mathcal{R}_{n,p}$.

We now prove statement (c). To this end, we first show that a deletion in $\mathbf{x}$ results in at most three deletions and two insertions in $\mathbb{1}_{\mathbf{w}}(\mathbf{x})$. Since $\mathbf{w}$ has length $2p$ and $\mathbb{1}_{\mathbf{w}}(\mathbf{x}) \in \mathcal{R}_{n,p}$ as shown in (a), a deletion in $\mathbf{x}$ results in at most two deletions and two insertions of 1 entries in $\mathbb{1}_{\mathbf{w}}(\mathbf{x})$, respectively. Otherwise, suppose that a deletion in $\mathbf{x}$ deletes three 1 entries $\mathbb{1}_{\mathbf{w}}(\mathbf{x})_{i_1}$, $\mathbb{1}_{\mathbf{w}}(\mathbf{x})_{i_2}$, and $\mathbb{1}_{\mathbf{w}}(\mathbf{x})_{i_3}$ in $\mathbb{1}_{\mathbf{w}}(\mathbf{x})$, then we have that $i_3 - i_1 \geq 2p$ because (a) holds. This is impossible since $\mathbf{w} \in \{0,1\}^{2p}$ and the deletion in $\mathbf{x}$ can not affect the two occurrences $\mathbf{x}_{i_1:i_1+2p-1}$ and $\mathbf{x}_{i_3:i_3+2p-1}$ of $\mathbf{w}$ in $\mathbf{x}$ simultaneously. Hence a deletion causes at most two deletions of 1 entries in $\mathbb{1}_{\mathbf{w}}(\mathbf{x})$ and similarly, the same holds for insertions.

Moreover, at most one 0 entry is deleted in $\mathbb{1}_{\mathbf{w}}(\mathbf{x})$ because of the deletion in $\mathbf{x}$. Hence, a deletion in $\mathbf{x}$ causes at most three deletions and two insertions in total in $\mathbb{1}_{\mathbf{w}}(\mathbf{x})$, and k deletions in $\mathbf{x}$ results in at most $3k$ deletions and $2k$ insertions in $\mathbb{1}_{\mathbf{w}}(\mathbf{x})$. The same holds for $\mathbf{y}$ and $\mathbb{1}_{\mathbf{w}}(\mathbf{y})$.

Since $\mathbf{x} \in \mathcal{B}_k(\mathbf{y})$, we conclude that $\mathbb{1}_{\mathbf{w}}(\mathbf{y})$ can be obtained from $\mathbb{1}_{\mathbf{w}}(\mathbf{x})$ by at most $5k$ deletions and $5k$ insertions, and hence, $\mathbb{1}_{\mathbf{w}}(\mathbf{x}) \in \mathcal{B}_{5k}(\mathbb{1}_{\mathbf{w}}(\mathbf{y}))$. □

With Lemma 7.2.2 and Lemma 7.2.5 established, we present a lower bound on the maximal value of polynomial $f_{\mathbf{x},\mathbf{w}}(z,0,\ldots,0) - f_{\mathbf{y},\mathbf{w}}(z,0,\ldots,0)$ for z close to 1. Note that it is important that z is located near the point 1 on the complex plane because of the scaling factor $(\frac{z-q}{1-q})^i$ in the multi-bit statistics in Eq. (7.3). To meet this requirement on z, existing works [15, 27, 37, 70] restrict z to lie on short subarcs of a unit circle around 1, a case also considered in [10] in the context of complex analysis. In this chapter, we choose z from a small circle around 1. It turns out that this choice of z achieves a lower bound $\frac{1}{n^{O(k)}}$ on $f_{\mathbf{x},\mathbf{w}}(z,0,\ldots,0) - f_{\mathbf{y},\mathbf{w}}(z,0,\ldots,0)$, which improves the bound $\frac{1}{n^{O(k^2)}}$ established in [37]. The details will be given in the following lemma, which is a critical result in this chapter. Its proof will be given in Sec. 7.4.

Lemma 7.2.6. *For integer $\ell \geq 1$ and strings $\mathbf{x}, \mathbf{y} \in \{0,1\}^n$ and $\mathbf{w} \in \{0,1\}^\ell$, if $\sum_{i=1}^n \mathbb{1}_{\mathbf{w}}(\mathbf{x})_i i^m \neq \sum_{i=1}^n \mathbb{1}_{\mathbf{w}}(\mathbf{y})_i i^m$ for some non-negative integer m, then there exists a complex number z, such that $|\frac{z-q}{1-q}|^n \leq 2$ and*

$$\sum_{i=1}^n \mathbb{1}_{\mathbf{w}}(\mathbf{x})_i z^i - \sum_{i=1}^n \mathbb{1}_{\mathbf{w}}(\mathbf{y})_i z^i \geq \frac{1}{n^{2m}(2m+2)} \tag{7.5}$$

for sufficiently large n.

Finally, we use the lower bound in Lemma 7.2.6 for single variate polynomial $f_{\mathbf{x},\mathbf{w}}(z,0,\ldots,0) - f_{\mathbf{y},\mathbf{w}}(z,0,\ldots,0)$ to obtain a lower bound for the multi-variate polynomial $f_{\mathbf{x},\mathbf{w}}(z_1,\ldots,z_\ell) - f_{\mathbf{y},\mathbf{w}}(z_1,\ldots,z_\ell)$, where $z_1,\ldots,z_\ell$ are close to 1. This lower bound guarantees a gap between the multi-bit statistics of $\tilde{X}$ and $\tilde{Y}$, which makes $\mathbf{x}$ and $\mathbf{y}$ distinguishable by Hoeffding's inequality (See Sec. 7.3). The proof follows similar steps to the ones in [15].

Lemma 7.2.7. *For integer $\ell \geq 1$ and strings $\mathbf{x}, \mathbf{y} \in \{0,1\}^n$ and $\mathbf{w} \in \{0,1\}^\ell$, if $\sum_{i=1}^n \mathbb{1}_{\mathbf{w}}(\mathbf{x})_i i^m \neq \sum_{i=1}^n \mathbb{1}_{\mathbf{w}}(\mathbf{y})_i i^m$ for some non-negative integer m, then there exist complex numbers $z_1,\ldots,z_\ell$, such that $|\frac{z_i-q}{1-q}|^n \leq 2$ for $j \in [\ell]$ and*

$$f_{\mathbf{x},\mathbf{w}}(z_1,\ldots,z_\ell) - f_{\mathbf{y},\mathbf{w}}(z_1,\ldots,z_\ell) \geq \frac{1}{n^{O(m)}} \tag{7.6}$$

for sufficiently large n.

Proof. According to Lemma 7.2.6, there exists a complex number z^* satisfying $|\frac{z^*-q}{1-q}|^n \leq 2$ and (7.5). Let $z_1 = z^*$ and $z_2 = \ldots = z_\ell = z$. Then the polynomial $f(z^*, z) \triangleq f_{\mathbf{x},\mathbf{w}}(z^*, z, \ldots, z) - f_{\mathbf{y},\mathbf{w}}(z^*, z, \ldots, z)$ is a function of z. By (7.3) and (7.5) we have that $f(z^*, 0) \geq \frac{1}{n^{2m}(2m+2)}$. The following result from [11] relates $f(z^*, 0)$ to the maximal value of $f(z^*, z)$ for z close to 1.

Proposition 7.2.2. *[11] Let $f(z)$ be an analytic function satisfying $f(z) \leq \frac{1}{1-|z|}$ for $|z| < 1$. There are positive real constants c_1 and c_2 such that*

$$|f(0)|^{\frac{c_1}{a}} \leq \exp(\frac{c_2}{a}) \max_{z\in[1-a,1]} |f(z)|$$

for real number $a \in (0,1]$.

According to Proposition 7.2.2, we have that

$$\begin{aligned}&\max_{z\in[\max\{2q-1,0\},1]} |f(z^*,z)|\\ \geq &\exp(-\frac{c_2}{1-\max\{2q-1,0\}})|f(z^*,0)|^{\frac{c_1}{1-\max\{2q-1,0\}}}\\ \geq &O(\frac{1}{n^{O(m)}}).\end{aligned} \tag{7.7}$$

Let $z_1 = z^*$ and $z_2 = \ldots = z_\ell$ be the number z maximizing the term $|f(z^*,z)|$ in (7.7). Then by Lemma 7.2.6 we have that $|\frac{z_1-q}{1-q}|^n \leq 2$ for sufficiently large n and $|\frac{z_i-q}{1-q}|^n \leq 1$ for $i \in \{2,\ldots,\ell\}$. Hence, the proof is done. □

7.3 Proof of Theorem 1

In this section we prove Theorem 1 based on the results from Lemma 7.2.1 to Lemma 7.2.5 and Lemma 7.2.7. Let t_0 be the smallest index such that $x_i \neq y_i$. If $t_0 < 12k$, we have the following result from [76], which was also used in [15].

Proposition 7.3.1. *For sequences $\mathbf{x}, \mathbf{y} \in \{0,1\}^n$, let t_0 be the smallest index such that $x_{t_0} \neq y_{t_0}$, i.e., $x_i = y_i$ for $i \in [t_0-1]$. Then, with high probability $\mathbf{x}$ and $\mathbf{y}$ can be distinguished using $\exp(O(t_0^{\frac{1}{3}}))$ samples.*

According to Proposition 7.3.1, sequences $\mathbf{x}$ and $\mathbf{y}$ can be distinguished with high probability using $\exp(O(t_0^{\frac{1}{3}})) < n^{O(k)}$ samples. Hence, it suffices to consider cases when $t_0 \geq 12k$.

Let $\mathbf{w}' = \mathbf{x}_{t_0-12k+1:t_0-1}$. By Lemma 7.2.4, either $(\mathbf{w}',0)$ or $(\mathbf{w}',1)$ is non-periodic. Without loss of generality, assume that $\mathbf{w} = (\mathbf{w}',0) \in \{0,1\}^{12k}$ is non-periodic. Then, similar to the arguments in [15, 27, 37, 57, 70], the core part of the proof is to show that the difference of multi-bit statistics $\mathbb{E}_{\tilde{X}}[\mathbb{1}_{\tilde{X}_{i_j}=w_j,\forall j\in[12k]}]$ and $\mathbb{E}_{\tilde{Y}}[\mathbb{1}_{\tilde{Y}_{i_j}=w_j,\forall j\in[12k]}]$, is at least $\frac{1}{n^{O(k)}}$ for some integers $1 \leq i_1 < \ldots < i_{12k} \leq n$, i.e.,

$$\max_{1\leq i_1<\ldots<i_{12k}\leq n} \Big|\mathbb{E}_{\tilde{X}}[\mathbb{1}_{\tilde{X}_{i_j}=w_j,\forall j\in[12k]}]$$

$$-\mathbb{E}_{\tilde{Y}}[\mathbb{1}_{\tilde{Y}_{i_j}=w_j,\forall j\in[12k]}]\Big| \geq \frac{1}{n^{O(k)}}. \tag{7.8}$$

Let

$$(i_1^*,\ldots,i_{12k}^*) = \operatorname{argmax}_{1\leq i_1<\ldots<i_{12k}\leq n}|\mathbb{E}_{\tilde{X}}[\mathbb{1}_{\tilde{X}_{i_j}=w_j,\forall j\in[12k]}] \\ -\mathbb{E}_{\tilde{Y}}[\mathbb{1}_{\tilde{Y}_{i_j}=w_j,\forall j\in[12k]}]|,$$

which can be determined once $\mathbf{x}$ and $\mathbf{y}$ are given. Suppose that $\mathbf{x}$ is passed through the deletion channel N times, generating N independent samples $\{\tilde{T}^t\}_{t=1}^N$. Then, by using similar Hoeffding's inequality (or the Chernoff bound) arguments as in [70], we can show that with high probability, the empirical distribution $\frac{\sum_{t=1}^N \mathbb{1}_{\tilde{T}^t_{i_j^*}=w_j,\forall j\in[12k]}}{N}$ is closer to $E\left[\mathbb{1}_{\tilde{X}_{i_j^*}=w_j,\forall j\in[12k]}\right]$ than to $E\left[\mathbb{1}_{\tilde{Y}_{i_j^*}=w_j,\forall j\in[12k]}\right]$, if

$$N \geq O\left(\frac{1}{|\mathbb{E}_{\tilde{X}}[\mathbb{1}_{\tilde{X}_{i_j^*}=w_j,\forall j\in[12k]}] - \mathbb{E}_{\tilde{Y}}[\mathbb{1}_{\tilde{Y}_{i_j^*}=w_j,\forall j\in[12k]}]|^2}\right) \\ = n^{O(k)}.$$

Hence $\mathbf{x}$ and $\mathbf{y}$ can be distinguished using $n^{O(k)}$ samples. Therefore, it suffices to show (7.8) in the rest of the proof.

Since $\mathbf{w}$ is non-periodic and $\mathbf{x} \in \mathcal{B}_k(\mathbf{y})$, Lemma 7.2.5 implies that $\mathbb{1}_{\mathbf{w}}(\mathbf{x}), \mathbb{1}_{\mathbf{w}}(\mathbf{y}) \in \mathcal{R}(n, 6k)$ and that $\mathbb{1}_{\mathbf{w}}(\mathbf{x}) \in \mathcal{B}_{5k}(\mathbb{1}_{\mathbf{w}}(\mathbf{y}))$. In addition, either $\mathbf{x}_{t_0-12k+1:t_0} = \mathbf{w}$ or $\mathbf{y}_{t_0-12k+1:t_0} = \mathbf{w}$ holds by definition of $\mathbf{w}$ and t_0. Therefore, we have that $\mathbb{1}_{\mathbf{w}}(\mathbf{x})_{t_0-12k+1} \neq \mathbb{1}_{\mathbf{w}}(\mathbf{y})_{t_0-12k+1}$, and thus that $\mathbb{1}_{\mathbf{w}}(\mathbf{x}) \neq \mathbb{1}_{\mathbf{w}}(\mathbf{y})$. Hence, we apply Lemma 7.2.2 and obtain an integer $m \in [12k+1]$ such that $\sum_{i=1}^n \mathbb{1}_{\mathbf{w}}(\mathbf{x})_i i^m \neq \sum_{i=1}^n \mathbb{1}_{\mathbf{w}}(\mathbf{x})_i i^m$. Then, according to Lemma 7.2.7, there exist complex numbers $z_1, \ldots, z_{12k}$, such that $|\frac{z_j-q}{1-q}|^n \leq 2$ for $j \in [12k]$ and (7.6) holds for sufficiently large n. Lemma 7.2.1 and Eq. (7.6) imply that

$$\sum_{1\leq i_1<\ldots<i_{12k}\leq n} \left|\mathbb{E}_{\tilde{X}}\left[\mathbb{1}_{\tilde{X}_{i_j}=w_j,\forall j\in[12k]}\right] - \mathbb{E}_{\tilde{Y}}\left[\mathbb{1}_{\tilde{Y}_{i_j}=w_j,\forall j\in[12k]}\right]\right| \\ (1-q)^{-12k}(\frac{z_1-q}{1-q})^{i_1}\prod_{j=2}^{12k}(\frac{z_j-q}{1-q})^{i_j-i_{j-1}-1} \\ \geq \frac{1}{n^{O(k)}},$$

and thus that

$$\max_{1\leq i_1<\ldots<i_{12k}\leq n} \left|\mathbb{E}_{\tilde{X}}\left[\mathbb{1}_{\tilde{X}_{i_j}=w_j,\forall j\in[12k]}\right] - \mathbb{E}_{\tilde{Y}}\left[\mathbb{1}_{\tilde{Y}_{i_j}=w_j,\forall j\in[12k]}\right]\right|$$

$$\geq \frac{1}{n^{O(k)}} \cdot (1-q)^{12k} \cdot \frac{1}{\binom{n}{12k}} \cdot \prod_{j=1}^{12k} \min\left\{1, |\frac{1-q}{z_j - q}|^n\right\}$$
$$= \frac{1}{n^{O(k)}}.$$

Therefore, (7.8) holds and the proof is done.

7.4 Proof of Lemma 7.2.6

Without loss of generality, assume that m is the smallest non-negative integer satisfying $\sum_{i=1}^{n} \mathbb{1}_{\mathbf{w}}(\mathbf{x})_i i^m \neq \sum_{i=1}^{n} \mathbb{1}_{\mathbf{w}}(\mathbf{x})_i i^m$. Let

$$f(z) = \sum_{i=1}^{n} \mathbb{1}_{\mathbf{w}}(\mathbf{x})_i z^i - \sum_{i=1}^{n} \mathbb{1}_{\mathbf{w}}(\mathbf{x})_i z^i \tag{7.9}$$

be a complex polynomial. The coefficients of $f(z)$ are within the set $\{-1, 0, 1\}$.

According to Lemma 7.2.3, we have that $f(z) = (z-1)^m q(x)$, where $q(z) = \sum_{i=0}^{n_1} c_i z^i$ is a complex polynomial with integer coefficients and $(z-1)$ does not divide $q(z)$, i.e., $q(1) \neq 0$. The following result was presented in [11]. It gives an upper bound on the norm of coefficients of $q(z)$.

Proposition 7.4.1. *[11] If a complex degree n polynomial $f(z)$ has all coefficients with norm not greater than 1, and can be factorized by*

$$f(z) = (z-1)^m q(z) = (z-1)^m (c_{n_1} z^{n_1} + \ldots + c_0),$$

then, we have that $\sum_{i=1}^{n_1} |c_i| \leq (n+1)(\frac{en}{m})^m$.

We are now ready to prove Lemma 7.2.6. Let $D \triangleq 2m+2$ and $z_j = \exp(\frac{2j\pi i}{D})$, $j \in [D]$ be a sequence of D complex numbers equally distributed on a unit circle. We first show that there exists a number $j \in [D]$ satisfying

$$q(1 + \frac{z_j}{n^2}) \geq \frac{1}{n^{O(m)}}.$$

Note that

$$\left|\sum_{j=1}^{D} q(1+\frac{z_j}{n^2})\right| = \left|\sum_{r=0}^{n_1} c_r \left[\sum_{j=1}^{D} (1 + \frac{\exp(\frac{2j\pi i}{D})}{n^2})^r\right]\right|$$
$$= \left|\sum_{r=0}^{n_1} c_r \sum_{s=0}^{r} \binom{r}{s} \sum_{j=1}^{D} \frac{\exp(\frac{2js\pi i}{D})}{n^{2s}}\right|$$

$$\begin{aligned}
&\overset{(a)}{=}\Big|\sum_{r=0}^{n_1} c_r \sum_{s=0}^{r}\binom{r}{s}\frac{D\mathbb{1}_{D\text{ divides }s}}{n^{2s}}\Big| \\
&=\Big|\sum_{r=0}^{n_1} c_r (D+\sum_{s=1}^{r}\binom{r}{s}\frac{D\mathbb{1}_{D\text{ divides }s}}{n^{2s}})\Big| \\
&=\Big|Dq(1)+\sum_{r=0}^{n_1}\sum_{s=1}^{r} c_r\binom{r}{s}\frac{D\mathbb{1}_{D\text{ divides }s}}{n^{2s}}\Big| \\
&\geq D|q(1)|-\sum_{s=1}^{n_1}(\sum_{r=s}^{n_1}|c_r|)\binom{n}{s}\frac{D\mathbb{1}_{D\text{ divides }s}}{n^{2s}} \\
&\overset{(b)}{\geq} D-(n+1)(\frac{en}{m})^m\sum_{s=1}^{n_1}\frac{D\mathbb{1}_{D\text{ divides }s}}{n^{s}} \\
&\geq D-D(n+1)(\frac{en}{m})^m\frac{1}{n^D}\sum_{t=0}^{\infty}\frac{1}{n^{Dt}} \\
&\geq D-D(n+1)(\frac{en}{m})^m\frac{2}{n^D} \\
&\overset{D\triangleq 2m+2}{=} D-D(n+1)(\frac{e}{mn})^m\frac{2}{n^2} \\
&=D-o(\frac{1}{n}) \\
&\geq 1
\end{aligned}$$

for sufficiently large n, where (a) follows from the identity

$$\sum_{j=1}^{D}\exp(\frac{2js\pi i}{D})=\frac{\exp(\frac{2sD\pi i}{D})-1}{\exp(\frac{2s\pi i}{D})-1}=D\mathbb{1}_{D\text{ divides }s}$$

and (b) follows from Proposition 7.4.1 and the facts that $q(1)$ is a nonzero integer and that $\binom{n}{s}\leq n^s$. Therefore, there exists an integer j such that

$$|q(1+\frac{z_j}{n^2})|\geq\frac{1}{D},$$

and thus that

$$\begin{aligned}
|f(1+\frac{z_j}{n^2})| &=\frac{|q(1+\frac{z_j}{n^2})|}{n^{2m}} \\
&\geq\frac{1}{n^{2m}(2m+2)}.
\end{aligned}$$

Moreover, we have that

$$|\frac{1+\frac{z_j}{n^2}-q}{1-q}|^n=\left(1+\frac{1}{n^4(1-q)^2}+2\frac{cos(\frac{2j\pi}{D})}{n^2(1-q)}\right)^{\frac{n}{2}}$$

$$\leq 2$$

for sufficiently large n. Hence, $z = 1 + \frac{z_j}{n^2}$ satisfies the conditions in Lemma 7.2.6.

7.5 Conclusion

In this chapter we studied the trace reconstruction problem when the string to be recovered is within bounded edit distance to a known string. Our result implies that when the edit distance is constant, the number of traces needed is polynomial. The problem of whether a polynomial number of samples suffices for the general trace reconstruction is open. However, it will be interesting to see if the methods in this chapter can be extended to obtain more general results.

Chapter 8

CODES FOR MULTI-HEAD RACETRACK MEMORIES

In this chapter, we consider error correction for racetrack memory applications. Though different from DNA-based storage. There are some similarities between models for racetrack memory and DNA storage, in the sense that both applications require correcting deletion/insertion errors, given multiple noisy copies of strings. The difference is that in DNA storage, the noisy copies are independent, given the string, while in racetrack memories, the noise in copies are correlated. The techniques in this chapter might in turn help in finding error correction schemes in DNA-storage, that jointly construct the encoders and decoders.

8.1 Introduction

Racetrack memory is a promising non-volatile memory that possesses the advantages of ultra-high storage density and low latency (comparable to SRAM latency) [75, 91]. It has a tape-like structure where the data is stored sequentially as a track of single-bit memory cells. The cells are accessed through read/write ports, called heads. When reading/writing the data, the heads stay fixed and the track is shifting.

One of the main challenges in developing racetrack memory systems is the limited precision in controlling the track shifts, that in turn affects the reliability of reading and writing the data [45, 100]. Specifically, the track may either not shift or shift more steps than expected. When the track does not shift, the same cell is read twice, causing a sticky insertion. When the track shifts more than a single step, cells are skipped, causing deletions in the reads [18].

It is natural to use deletion and sticky insertion correcting codes to deal with shift errors. Also, it is known that a code correcting k deletions is capable of correcting s deletions and r insertions when $s+r \leq k$ [64]. However, designing redundancy and complexity efficient deletion correcting codes has been an open problem for decades, though there is a significant advance toward the solution recently. In fact, no deletion correcting codes with rate approaching 1 were known until [12] proposed a code with redundancy $128k^2 \log k \log n + o(\log n)$. After [12], the work of [22] and [86] independently proposed k deletion codes with $O(k \log n)$ bits of redundancy, which are order-wise optimal. Following [86], [87] proposed a systematic deletion code with $4k \log n + o(\log n)$ bits of redundancy and is computationally efficient for constant k. The redundancy was later improved in [90] to $(4k - 1) \log n + o(\log n)$. Evidently, for k, a constant number of deletions, the redundancy of this code is orders of magnitude away from optimal, known to be in the range $k \log n + o(\log n)$ to $2k \log n + o(\log n)$ [64]. Hence, it is natural to explore constructions of deletion and insertion correcting codes that are specialized for racetrack memories and might provide more efficient redundancy and lower complexity encoding/decoding algorithms.

There are two approaches for construction of codes for racetrack memories. The first is to leverage the fact that there are multiple parallel tracks with a single head per-track, and the second, is to add redundant heads per-track. For the multiple parallel head structure, the proposed codes in [94] can correct up to two deletions per head and the proposed codes in [17] can correct l bursts of deletions, each of length at most b. The codes in [17] are asymptotically (in the number of heads) rate-optimal. The second approach for combating deletions in racetrack memories is to use redundant heads per-track [100, 18, 19]. As shown in Fig. 8.1, a track is read by multiple heads, resulting in multiple copies (potentially erroneous) of the same sequence. This can be regarded as a sequence reconstruction problem, where a sequence $\mathbf{c}$ needs to be recovered from multiple copies, each obtained after k deletions in $\mathbf{c}$. We emphasize that the general sequence reconstruction problem [67] is different from the current settings, as here the heads are at fixed and known positions, hence, the set of deletion locations in one head is a shift of that in another head [18]. This is because the heads stay fixed and thus the deletion locations in their reads have fixed relative distances. Demonstrating the advantage of multiple heads, the paper [19] proposed an efficient k-deletion code of length n with redundancy $\log \log n + 4$ and an efficiently $(k - 1)$-deletion code with $O(1)$ bits of redundancy, using k heads. In contrast, for general k-deletion codes, the lower bound on the redundancy is $k \log n$. However, the code in [19] is required to use d heads and is limiting k to be smaller or equal to d. It is known that the number of heads affects the area overhead of the racetrack memory device [18], hence, it motivates the following **natural question**: What is the best redundancy that can be achieved for a k-deletion code (k is a constant) if the number of heads is fixed at d (due to area limitations)?

One of our **key results** is an answer to this question, namely, we construct codes that can correct k deletions, for any k beyond the known limit of d. Our code

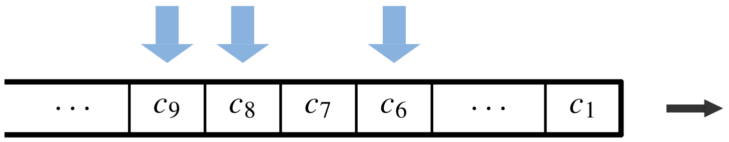

Figure 8.1: Racetrack memory with multiple heads.

has $O(k^4 d \log\log n)$ redundancy for the case when $k \leq 2d-1$. In addition, when $k \geq 2d$, the code has $2\lfloor k/d \rfloor \log n + o(\log n)$ redundancy. Our key result is summarized formally by the following theorem. Notice that the theorem implies that the redundancy of our codes is asymptotically larger than optimal by a scale factor of at most four.

Theorem 8.1.1. *For a constant integer k, let the distance t_i between the i-th and $(i+1)$-th heads be $t_i \geq \max\{(3k+\lceil\log n\rceil+2)[k(k-1)/2+1]+(7k-k^3)/6, (4k+1)(5k+\lceil\log n\rceil+3)\}$ for $i \in \{1,\ldots,d-1\}$. Then for $d \leq k \leq 2d-1$, there exists a length $N = n + 4k\log\log n + o(\log\log n)$ d head k-deletion correcting code with redundancy $4k\log\log n + o(\log\log n)$. For $k \geq 2d$, there exists a length $N = n + 2\lfloor k/d\rfloor \log n + o(\log n)$ d head k-deletion correcting code with redundancy $2\lfloor k/d\rfloor \log n + o(\log n)$. The encoding and decoding functions can be computed in n^{2k+1} time. Moreover, for $k \geq 2d$ and $t_i = n^{o(1)}$, the amount of redundancy of a d head k-deletion correcting code is lower bounded by $\lfloor k/2d\rfloor \log n + o(\log n)$.*

Since in addition to deletion errors, sticky insertion errors and substitution errors occur in racetrack memories, we are interested in codes that correct not only deletions, but a combination of deletion, sticky insertion, and substitution errors in multi-head racetrack memories. However, contrast to the single-head cases where a deletion code is also a deletion/insertion code, there is no such equivalence in multi-head racetrack memories. Correcting a combination of at most k deletions and sticky insertions in total turns out to be more difficult than correcting k deletion errors. It is not known whether the k-deletion code with $\log\log n + O(1)$ redundancy and the $(k-1)$-deletion code with $O(1)$ redundancy in [18] extend to a combination of deletion and sticky insertion errors in a k head racetrack memory.

Our second result, which is the main result in this chapter, provides an answer for such scenarios. Moreover, we consider a more general problem of correcting a combination of deletions and insertions in a d-head racetrack memory, rather than deletions and sticky insertions, and show that the redundancy result for deletion cases extends to cases with a combination of deletions and insertions. Note that this covers the cases with deletion, insertion, and substitution errors, since a substitution is a deletion followed by an insertion.

Theorem 8.1.2. *For a constant integer k, let the distance t_i between the i-th and $(i+1)$-th heads be equal and $t_i = t > (\frac{k^2}{4}+3k+2)(6k+\lceil\log n\rceil+3)+8k+\lceil\log n\rceil+3$ for $i \in \{1,\ldots,d-1\}$. Then for $k < d$, there exists a length $N = n+k+1+O(1)$ code correcting a combination of at most k insertions and deletions in a d head racetrack memory with redundancy $k+1+O(1)$. The encoding and decoding complexity is $poly(n)$. For $d \leq k \leq 2d-1$, there exists a length $N = n+4k\log\log n+o(\log\log n)$ code correcting a combination of at most k insertions and deletions in a d head racetrack memory with redundancy $4k\log\log n + o(\log\log n)$. Finally, when $d \geq 2d$, there exists a length $N = n + 2\lfloor k/d\rfloor \log n + o(\log n)$ code that corrects a combination of at most k insertions and deletions in a d head racetrack memory with redundancy $2\lfloor k/d\rfloor \log n+o(\log n)$. The encoding and decoding functions can be computed in n^{2k+1} time.*

Remark 8.1.1. *Theorem 8.1.2 improves the head distance in Theorem 8.1.1 when k is sufficiently large.*

Organization: In Sec. 8.2, we present the problem settings and some basic lemmas needed in our proof. Sec. 8.3 presents the proof of the main result for the case $k \leq 2d-1$. Sec. 8.4 describes in detail how to synchronize the reads. The case $k \geq 2d$ is addressed in Sec. 8.5. Sec. 8.7 concludes the chapter.

8.2 Preliminaries

Problem Settings

We now describe the problem settings and the notations needed. For any two integers $i \leq j$, let $[i,j] = \{i, i+1, \ldots, j-1, j\}$ be an integer interval that contains all integers between i and j. Let $[i,j] = \emptyset$ for $i > j$. For a length n sequence $\mathbf{c} = (c_1,\ldots,c_n)$, an index set $\mathcal{I} \subseteq [1,n]$, let

$$\mathbf{c}_{\mathcal{I}} = (c_i : i \in \mathcal{I})$$

be a subsequence of $\mathbf{c}$, obtained by choosing bits with locations in the location set $\mathcal{I}$. Denote by $\mathcal{I}^c = [1,n]\backslash\delta$ the complement of $\mathcal{I}$.

In the channel model of a d head racetrack memory, the input is a binary sequence $\mathbf{c} \in \{0,1\}^n$. The channel output consists of d subsequences of $\mathbf{c}$ of length $n-k$, obtained by the d heads after k deletions in the channel input $\mathbf{c}$, respectively. Each subsequence is called a *read*. Let $\boldsymbol{\delta}_i = \{\delta_{i,1}, \ldots, \delta_{i,k}\} \subseteq [1,n]$ be the deletion locations of the i-th head such that $\delta_{i,1} < \ldots < \delta_{i,k}$. Then, the read from the i-th head is given by $\mathbf{c}_{\boldsymbol{\delta}_i^c}$, $i \in [1,d]$, i.e., bits c_ℓ, $i \in \boldsymbol{\delta}_\ell$ are deleted.

Note that in a d-head racetrack memory, the heads are placed in fixed positions, and the deletions are caused by "over-shifts" of the track. Hence when a deletion occurs at the j-th bit in the read of the i_1-th head, a deletion also occurs at the $(j+t')$-th bit in the read of the i_2-th head, where t' is the distance between the i_1-th head and the i_2-th head. Let t_i be the distance between the i-th head and the $i+1$-th head, $i \in [1, d-1]$. Then, the deletion location sets $\{\boldsymbol{\delta}_i\}_{i=1}^d$ satisfy

$$\boldsymbol{\delta}_{i+1} = \boldsymbol{\delta}_i + t_i,$$

for some positive integer t_i, $i \in [1, d-1]$, where for an integer set $\mathcal{S}$ and an integer t, $\mathcal{S} + t = \{x + t : x \in \mathcal{S}\}$.

To formally define a code for d-head racetrack memories, we represent the d reads from the d heads by a $d \times (n-k)$ binary matrix, called the *read matrix*. The i-th row of the read matrix is the read from the i-th head. Let $\boldsymbol{D}(\mathbf{c}, \boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_d) \in \{0,1\}^{d\times(n-k)}$ be the read matrix of a d head racetrack memory, where the input is $\mathbf{c} \in \{0,1\}^n$ and the deletion locations in the i-th head are given by $\boldsymbol{\delta}_i$, $i \in [1,d]$. By this definition, the i-th row of $\boldsymbol{D}(\mathbf{c}, \boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_d)$ is $\mathbf{c}_{\boldsymbol{\delta}_i^c}$.

Example 8.2.1. *Consider a* 3*-head racetrack memory with head distance* $t_1 = 1$ *and* $t_2 = 2$*. Let the deletion location set* $\boldsymbol{\delta}_1 = \{2,5,7\}$*. Then, we have that* $\boldsymbol{\delta}_2 = \{3,6,8\}$ *and* $\boldsymbol{\delta}_3 = \{5,8,10\}$*. Let* $\mathbf{c} = (1,1,0,1,0,0,0,1,0,1)$ *be a sequence of length* 10*. Then, the read matrix is given by*

$$\boldsymbol{D}(\mathbf{c}, \boldsymbol{\delta}_1, \boldsymbol{\delta}_2, \boldsymbol{\delta}_3) = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

The deletion ball $\mathcal{D}_k(\mathbf{c}, t_1, \ldots, t_{d-1})$ of a sequence $\mathbf{c} \in \{0,1\}^n$ is the set of all possible read matrices in an d head racetrack memory with input $\mathbf{c}$ and head distance t_i, $i \in [1, d-1]$, i.e.,

$$\mathcal{D}_k(\mathbf{c}, t_1, \ldots, t_{d-1}) = \{\boldsymbol{D}(\mathbf{c}, \boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_d) : \boldsymbol{\delta}_{i+1} = \boldsymbol{\delta}_i + t_i, \boldsymbol{\delta}_i \subseteq [1,n], |\boldsymbol{\delta}_i| = k, \\ i \in [1, d-1]\}.$$

A d-head k-deletion code C is the set of all sequences such that the deletion balls of any two do not intersect, i.e., for any $\mathbf{c}, \mathbf{c}' \in C$, $\mathcal{D}_k(\mathbf{c}, t_1, \ldots, t_{d-1}) \cap \mathcal{D}_k(\mathbf{c}', t_1, \ldots, t_{d-1}) = \emptyset$.

The following notations will be used throughout this chapter. For a matrix $\boldsymbol{A}$ and two index sets $\mathcal{I}_1 \subseteq [1, d]$ and $\mathcal{I}_2 \subseteq [1, n-k]$, let $\boldsymbol{A}_{\mathcal{I}_1, \mathcal{I}_2}$ denote the submatrix of $\boldsymbol{A}$ obtained by selecting the rows $i \in \mathcal{I}_1$ and the columns $j \in \mathcal{I}_2$. For any two integer sets $\mathcal{S}_1$ and $\mathcal{S}_2$, the set $\mathcal{S}_1 \backslash \mathcal{S}_2 = \{x : x \in \mathcal{S}_1, x \notin \mathcal{S}_2\}$ denotes the difference of sets $\mathcal{S}_1$ and $\mathcal{S}_2$.

A sequence $\mathbf{c} \in \{0, 1\}^n$ is said to have period ℓ if $c_i = c_{i+\ell}$ for $i \in [1, n-\ell]$. We use $L(\mathbf{c}, i)$ to denote the length of the longest subsequence of consecutive bits in $\mathbf{c}$ that has period i. Furthermore, define

$$L(\mathbf{c}, \leq k) \triangleq \max_{i \leq k} L(\mathbf{c}, i).$$

Example 8.2.2. *Let the sequence* $\mathbf{c}$ *be* $\mathbf{c} = (1, 1, 0, 1, 1, 0, 1, 0, 0)$*. Then we have that* $L(\mathbf{c}, 1) = 2$*,* $L(\mathbf{c}, 2) = 4$*, and that* $L(\mathbf{c}, 3) = 7$*. Thus, we have* $L(\mathbf{c}, \leq 3) = 7$*.*

Racetrack Memory with Insertion and Deletion Errors

We now describe the notations and problem settings for d-head racetrack memories with a combination of insertion and deletion errors, which is similar to d-head racetrack memories with deletion errors only. In addition to the deletion errors described by deletion location sets $\{\boldsymbol{\delta}_i\}_{i=1}^d$, we consider insertion errors described by insertion location sets $\{\gamma_i\}_{i=1}^d$ satisfying

$$\boldsymbol{\delta}_{i+1} = \boldsymbol{\delta}_i + t_i,$$

where $\gamma_i = \{\gamma_{i,1}, \ldots, \gamma_{i,s}\}$ for $i \in [1, d]$, and the inserted bits $\mathbf{b}_i = b_{i,1}, b_{i,2}, \ldots, b_{i,s}$. It is assumed that $\gamma_{i,j} \in [0, n]$ for $i \in [1, d]$ and $j \in [1, n]$. As a result of the insertion errors, bit $b_{i,j}$ is inserted right after the $\gamma_{i,j}$-th bit of $\mathbf{c}$ in the i-th head, for $i \in [1, d]$ and $j \in [1, n]$. when $\gamma_{i,j} = 0$, the insertion occurs before c_1. We note that $\mathbf{b}_i$ can be different for different i's.

We call a deletion error or an insertion error an *edit error*, or *error*, in Sec. 8.6. For edit errors, define the read matrix $\boldsymbol{E}(\mathbf{c}, \boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_d, \gamma_1, \ldots, \gamma_d, \mathbf{b}_1, \ldots, \mathbf{b}_d) \in \{0, 1\}^{d \times (n+s-r)}$, where $|\boldsymbol{\delta}_i| = s$ for $i \in [1, d]$, as follows. The i-th row of

$$\boldsymbol{E}(\mathbf{c}, \boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_d, \gamma_1, \ldots, \gamma_d, \mathbf{b}_1, \ldots, \mathbf{b}_d) \in \{0, 1\}^{d \times (n+s-r)}$$

is obtained by deleting the bits $c_{\ell : \ell \in \boldsymbol{\delta}_i}$ and inserting $b_{i,j}$ after the $c_{\gamma_{i,j}}$, for $i \in [1, d]$ and $j \in [1, s]$. In this chapter, we consider k edit errors. Hence, $r + s \leq k$.

Example 8.2.3. *Continue Example 8.2.1. Consider a* 3*-head racetrack memory with head distance* $t_1 = 1$ *and* $t_2 = 2$*. Let the deletion location set* $\boldsymbol{\delta}_1 = \{2, 5, 7\}$*. Then, we have that* $\boldsymbol{\delta}_2 = \{3, 6, 8\}$ *and* $\boldsymbol{\delta}_3 = \{5, 8, 10\}$*. In addition, the insertion location set is given by* $\gamma_1 = \{0, 2\}$*. Then, we have* $\gamma_2 = \{1, 3\}$*, and* $\gamma_3 = \{3, 5\}$*. Let* $\mathbf{b}_1 = (1, 1)$*,* $\mathbf{b}_2 = (1, 0)$*,* $\mathbf{b}_3 = (0, 1)$ *Let* $\mathbf{c} = (1, 1, 0, 1, 0, 0, 0, 1, 0, 1)$ *be a sequence of length* 10*. Then, the read matrix is given by*

$$\boldsymbol{E}(\mathbf{c}, \boldsymbol{\delta}_1, \boldsymbol{\delta}_2, \boldsymbol{\delta}_3, \gamma_1, \gamma_2, \gamma_3, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3) = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

Define an edit ball $\mathcal{E}_k(\mathbf{c}, t_1, \ldots, t_{d-1})$ of a sequence $\mathbf{c} \in \{0, 1\}^n$ as the set of all possible read matrices in an d head racetrack memory with input $\mathbf{c}$ and head distance t_i, $i \in [1, d-1]$, i.e.,

$$\begin{aligned}&\mathcal{E}_k(\mathbf{c}, t_1, \ldots, t_{d-1}) = \{\boldsymbol{E}(\mathbf{c}, \mathbf{c}, \boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_d, \gamma_1, \ldots, \gamma_d, \mathbf{b}_1, \ldots, \mathbf{b}_d) : \boldsymbol{\delta}_{i+1} = \boldsymbol{\delta}_i + t_i,\\ &\gamma_{i+1} = \gamma_i + t_i, \text{ for } i \in [1, d], \text{and} \boldsymbol{\delta}_i \subseteq [1, n], |\boldsymbol{\delta}_i| = r, \gamma_i \subseteq [0, n], |\gamma_i| = s,\\ &\mathbf{b}_i \in \{0, 1\}^s \text{ for } i \in [1, d], r + s \leq k, \}.\end{aligned}$$

A d-head k-edit correction code C is the set of all sequences such that the edit balls of any two do not intersect, i.e., for any $\mathbf{c}, \mathbf{c}' \in C$, $\mathcal{E}_k(\mathbf{c}, t_1, \ldots, t_{d-1}) \cap \mathcal{E}_k(\mathbf{c}', t_1, \ldots, t_{d-1}) = \emptyset$.

Lemmas

In this section we present lemmas that will be used throughout this chapter. Some of them are existing results. The following lemma describes a systematic Reed-Solomon code that can correct a constant number of erasures and can be efficiently computed (See for example [97]).

Lemma 8.2.1. *Let* k*,* q*, and* n *be integers that satisfy* $n + k \leq q$*. Then, there exists a map* $RS_k : \mathbb{F}_q^n \to \mathbb{F}_q^k$*, computable in* $poly(n)$ *time, such that* $\{(\mathbf{c}, RS_k(\mathbf{c})) : \mathbf{c} \in \mathbb{F}_q^n\}$ *is a* k*-erasure correcting code.*

The Reed-Solomon code requires $O(\log n)$ redundancy for correcting k erasures. Correcting a burst of two erasures requires less redundancy when the alphabet size of the code has order $o(\log n)$. The following code for correcting consecutive two erasures will be used for the case when the number of deletions k is less than two times the number of heads m.

Lemma 8.2.2. *For any integers n and q, there exists a map $ER : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^2$, computable in $O(n)$ time, such that the code $\{(\mathbf{c}, ER(\mathbf{c}) : \mathbf{c} \in \mathbb{F}_q^n\}$ is capable of correcting two consecutive erasures.*

Proof. For a sequence $\mathbf{c} = (c_1, \ldots, c_n)$ Let the code ER be given by

$$ER(\mathbf{c}) = (\sum_{i=0}^{\lfloor (n-1)/2 \rfloor} c_{2i+1}, \sum_{i=0}^{\lfloor n/2 \rfloor} c_{2i}),$$

which are the sums of symbols with odd and even indices respectively over field $\mathbb{F}_q$. Note that a consecutive of two erasures is reduces to single erasures in the even symbols and odd symbols respectively, which can be recovered with the help of $ER(\mathbf{c})$. Hence, $(\mathbf{c}, ER(\mathbf{c}))$ can be recovered from two consecutive erasures. □

Our construction is based on a systematic deletion code for a single read $d = 1$, which was presented in Ch. 4.

Lemma 8.2.3. *Let k be a fixed integer. For integers B and n. There exists a hash function*

$$Hash : \{0, 1\}^B \rightarrow \{0, 1\}^{\lceil 4k \log B + o(\log B)}$$

computable in $O(B^{2k+1})$ time, such that any sequence $\mathbf{c} \in \{0, 1\}^B$ can be recovered from its length $B - k$ subsequence with the help of of $Hash(\mathbf{c})$.

We also use the following fact, proved in [64], which implies that a deletion correcting code can be used to correct a combination of deletions and insertions.

Lemma 8.2.4. *A k-deletion correcting code is capable of correcting a combination of r deletions and s insertions, where $r + s \leq k$.*

Remark 8.2.1. *Note that the lemma does not hold in a multi-head racetrack memory considered in this chapter.*

In addition, in order to synchronize the sequence $\mathbf{c}$ in the presence of deletions, we need to transform $\mathbf{c}$ to a sequence that has a limited length constraint on its periodic subsequences. Such constraint was used in [18], where it was proved that the redundancy of the code $\{\mathbf{c} : L(\mathbf{c}, \leq k) \leq \lceil \log n \rceil + k + 1\}$ is at most 1 bit. In the following lemma we present a method to transform any sequence to one that satisfies this constraint. The redundancy of our construction is $k + 1$ bits. However, it is small compared to the redundancy of the d head k-deletion code.

Lemma 8.2.5. *For any integers k and n, there exists an injective function $F : \{0,1\}^n \to \{0,1\}^{n+k+1}$, computable in $O_k(n^3 \log n)$ time, such that for any sequence $\{0,1\}^n$, we have that $L(F(\mathbf{c}), \leq k) \leq 3k+2+\lceil \log n \rceil$.*

Proof. Let $\mathbf{1}^x$ and $\mathbf{0}^y$ denote consecutive x 1's and consecutive y 0's respectively. The encoding procedure for computing $F(\mathbf{c})$ is as follows.

1. **Initialization:** Let $F(\mathbf{c}) = \mathbf{c}$. Append $(\mathbf{1}^k, 0)$ to the end of the sequence $F(\mathbf{c})$. Let $i = 1$ and $n' = n$. Go to Step 1.
2. **Step 1:** If $i \leq n' - 2k - \lceil \log n \rceil - 1$ and $F(\mathbf{c})_{[i,i+2k+\lceil \log n \rceil+1]}$ has period $p \leq k$, let p_{min} be the smallest period of $F(\mathbf{c})_{[i,i+2k+\lceil \log n \rceil+1]}$. Delete $F(\mathbf{c})_{[i,i+2k+\lceil \log n \rceil+1]}$ from $F(\mathbf{c})$ and append $(\mathbf{1}^{k-p_{min}}, 0, F(\mathbf{c})_{[i,i+p_{min}-1]}, i, \mathbf{0}^{k+1})$ to the end of $F(\mathbf{c})$, i.e., set $F(\mathbf{c})_{[i,n-k-\lceil \log n \rceil-1]} = F(\mathbf{c})_{[i+2k+\lceil \log n \rceil+2,n+k+1]}$ and $F(\mathbf{c})_{[n-k-\lceil \log n \rceil,n+k+1]} = (\mathbf{1}^{k-p_{min}}, 0, F(\mathbf{c})_{[i,i+p_{min}-1]}, i, \mathbf{0}^{k+1})$. Let $n' = n' - 2k - \lceil \log n \rceil - 2$ and $i = 1$. Repeat. Else go to Step 2.
3. **Step 2:** If $i \leq n' - 2k - \lceil \log n \rceil - 1$, let $i = i+1$ and go to Step 1. Else output $F(\mathbf{c})$.

It can be verified that the length of the sequence $F(\mathbf{c})$ remains to be $n+k+1$ during the procedure. The number n' in the procedure denotes the number such that $F(\mathbf{c})_{[n'+1,n+k+1]}$ are appended bits and $F(\mathbf{c})_{[1,n']}$ are the remaining bits in $\mathbf{c}$ after deletions. Since either i increases to n' or n' decreases in Step 1. The algorithm terminates within $O(n^2)$ times of Step 1 and Step 2. Since it takes $O(k(3k+2+\log n)n)$ to check the periodicity in Step 1. The total complexity is $O_k(n^3 \log n)$.

We now prove that $L(F(\mathbf{c}), \leq k) \leq 3k+2+\lceil \log n \rceil$. Let n' be the number computed in the encoding procedure. According to the encoding procedure, we have that $L(F(\mathbf{c})_{[j,j+2k+1+\lceil \log n \rceil]}, \leq k) \leq 2k+1+\lceil \log n \rceil$ for $j \leq n' - 2k - \lceil \log n \rceil - 1$, since any subsequence $F(\mathbf{c})_{[j,j+2k+1+\lceil \log n \rceil]}$ with period not greater than k is deleted. Therefore $L(F(\mathbf{c})_{[j,j+3k+1+\lceil \log n \rceil]}, \leq k) \leq 3k+2+\lceil \log n \rceil$ for $j \leq n' - 2k - \lceil \log n \rceil - 1$. For $n' - 2k - \lceil \log n \rceil \leq j \leq n'$, the sequence $F(\mathbf{c})_{[j,j+2k+1+\lceil \log n \rceil]}$ contains $F(\mathbf{c})_{[n'+1,n'+k+1]} = (\mathbf{1}^k, 0)$, which does not have period not greater than k. Hence we have that $L(F(\mathbf{c})_{[j,j+3k+1+\lceil \log n \rceil]}, \leq k) \leq 3k+2+\lceil \log n \rceil$. For $j > n'$, the sequence $F(\mathbf{c})_{[j,j+3k+1+\lceil \log n \rceil]}$ contains $\mathbf{0}^{k+1}$ as $k+1$ consecutive bits. Hence, if $F(\mathbf{c})_{[j,j+3k+1+\lceil \log n \rceil]} = 3k+2+\lceil \log n \rceil$, we have that $F(\mathbf{c})_{[j,j+3k+1+\lceil \log n \rceil]} = \mathbf{0}^{3k+2+\lceil \log n \rceil}$. However, this is impossible since $F(\mathbf{c})_{[j,j+3k+1+\lceil \log n \rceil]}$ contains either the location index i to the left of $\mathbf{0}^{k+1}$ or the bits $(\mathbf{1}^{k-p_{min}}, 0, F(\mathbf{c})_{[i,i+p_{min}-1]})$ to the right of $\mathbf{0}^{k+1}$, both of which can not be all zero. Therefore, we conclude that $L(\mathbf{c}, \leq k) \leq 3k + 2 + \lceil \log n \rceil$. Given $F(\mathbf{c})$, the decoding procedure for computing $\mathbf{c}$ is as follows.

1. **Initialization:** Let $\mathbf{c} = F(\mathbf{c})$ and go to Step 1.
2. **Step 1:** If $\mathbf{c}_{[n+1,n+k+1]} \neq (\mathbf{1}^k, 0)$, let j be the length of the first 1 run in $\mathbf{c}_{[n-k-\lceil \log n \rceil, n+k+1]}$ and let p be the decimal representation of $\mathbf{c}_{n-\lceil \log n \rceil+1,n}$. Let $\mathbf{a}$ be a sequence of length $2k + \lceil \log n \rceil + 2$ and period $k - j$. The first $k - j$ bits of $\mathbf{a}$ is given by $\mathbf{c}_{[n-k-\lceil \log n \rceil + j+1, n-\lceil \log n \rceil]}$. Delete $\mathbf{c}_{[n-k-\lceil \log n \rceil, n+k+1]}$ from $\mathbf{c}$ and insert $\mathbf{a}$ at location p of $\mathbf{c}$, i.e., let $\mathbf{c}_{[p+2k+\lceil \log n \rceil+2, n+k+1]} = \mathbf{c}_{[p, n-k-\lceil \log n \rceil - 1]}$ and $\mathbf{c}_{[p, p+2k+\lceil \log n \rceil+1]} = \mathbf{a}$. Repeat. Else output $\mathbf{c}$.

Note that the encoding procedure consists of a series of deleting and appending operations. The decoding procedure consists of a series of deletion and inserting operations. Let $F_i(\mathbf{c})$, $i \in [0, R]$ be the sequence $F(\mathbf{c})$ obtained after the i-th deleting and appending operation in the encoding procedure, where R is the number of deleting and appending operations in total in the encoding procedure. We have that $F_0(\mathbf{c}) = \mathbf{c}$ and $F_R(\mathbf{c})$ is the final output $F(\mathbf{c})$. It can be seen that the decoding procedure obtains $F_{R-i}(\mathbf{c})$, $i \in [0, R]$ after the i-th deleting and inserting operation. Hence the function $F(\mathbf{c})$ is injective. □

Finally, we restate one of the main results in [18] that will be used in our construction. The result guarantees a procedure to correct $d - 1$ deletions in a d head racetrack memory, given that the distance between consecutive heads are large enough.

Lemma 8.2.6. *Let $d \leq k$ be two integers and C be a $k - d + 1$ deletion code, then $C \cap \{\mathbf{c} : L(\mathbf{c}, \leq k) \leq T\}$ is an d head k deletion correcting code, given that the distance between consecutive heads $t \geq T[k(k-1)/2 + 1] + (7k - k^3)/6$ for $i \in [1, d-1]$.*

8.3 Correcting up to $2d - 1$ Deletions with d Heads

In this section we construct a k deletion d head code for cases when $k \leq 2d - 1$. To this end, we first present a lemma that is crucial in our code construction. The lemma states that the range of deletion locations can be narrowed down to a list of short intervals. Moreover, the number of deletions within each interval can be determined. The proof of the lemma will be given in Sec. 8.4. Before presenting the

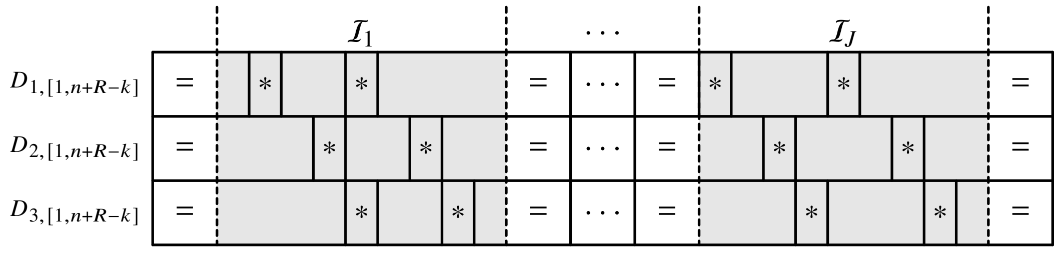

Figure 8.2: An illustration of Lemma 8.3.1. The $*$ entries denote deletion in the heads. The read $D_{i,[1,n+R]}$ in each head is obtained after deleting the $*$ entries from $\mathbf{c}$.

lemma, we give the following definition, which describes a property of the intervals we look for.

Definition 8.3.1. *Let $\boldsymbol{\delta}_i = \{\delta_{i,1}, \ldots, \delta_{i,k}\}$ be the set of deletion locations in the i-th head of a d head racetrack memory, i.e., $\boldsymbol{\delta}_{i+1} = \boldsymbol{\delta}_i + t_i$, for $i \in [1, d-1]$. An interval $\mathcal{I}$ is* deletion isolated *if*

$$\boldsymbol{\delta}_{i+1} \cap \mathcal{I} = t_i + \boldsymbol{\delta}_i \cap \mathcal{I},$$

for $i \in [1, d-1]$.

Example 8.3.1. *Consider a 3-head racetrack memory with head distances $t_1 = 1$ and $t_2 = 2$. Let the length of the sequence $\mathbf{c}$ be $n = 22$ and the deletion positions in three heads be given by*

$$\boldsymbol{\delta}_1 = \{1, 2, 4, 8, 14, 17\},$$
$$\boldsymbol{\delta}_2 = \{2, 3, 5, 9, 15, 18\}, \text{ and}$$
$$\boldsymbol{\delta}_3 = \{4, 5, 7, 11, 17, 20\}.$$

Then the intervals $[1, 7]$, $[8, 12]$, and $[14, 22]$ are all deletion isolated.

Intuitively, an interval $\mathcal{I}$ is deletion isolated when the subsequences $\mathbf{c}_{\mathcal{I} \cap \delta_i^c}$ for $i \in [1, d]$ can be regarded as the d reads of the sequence $\mathbf{c}_{\mathcal{I}}$ in a d-head racetrack memory after $|\boldsymbol{\delta}_1 \cap \mathcal{I}|$ deletions in each head.

Lemma 8.3.1. *For any positive integers n and $R \geq k + 1$, let $\mathbf{c} \in \{0, 1\}^{n+R}$ be a sequence such that $L(\mathbf{c}_{[1,n+k+1]}, \leq k) \leq 3k + \lceil \log n \rceil + 2 \triangleq T$. Let the distance t_i*

between head i and head $i+1$ satisfy $t_i \geq (4k+1)(T+2k+1) \triangleq T_{min}$, $i \in [1, d-1]$,k and $t_{max} = \max_{i\in\{1,\dots,d-1\}} t_i$ be the largest distance between two consecutive heads. Then given $\boldsymbol{D} \in \mathcal{D}_k(\mathbf{c}, t_1, \dots, t_{d-1})$, it is possible to find a set of $J \leq k$ disjoint and deletion isolated intervals $\mathcal{I}_j \subseteq [1, n+R]$, $j \in [i, J]$ such that $\boldsymbol{\delta}_w \subset \cup_{j=1}^{J} \mathcal{I}_j$ for $w \in [1, d]$ and

$$|\mathcal{I}_j \cap [1, n+k+1]| \leq (2\lfloor(2t_{max}+T+1)/2\rfloor+1)kd + \lfloor(2t_{max}+T+1)/2\rfloor + k \triangleq B,$$

for $j \in [1, J]$. Moreover, $|\boldsymbol{\delta}_1 \cap \mathcal{I}_j|$ can be determined for $j \in [1, J]$.

An illustration of Lemma 8.3.1 is shown in Fig. 8.2. Since the interval $\mathcal{I}_j$ is deletion isolated for $j \in [1, J]$, all rows of $\boldsymbol{D}$ are aligned in locations $[1, n]\backslash(\cup_{j=1}^{J}\mathcal{I}_j)$, i.e., the entries in the i-th column of A correspond to the same bit in $\mathbf{c}$ for $i \in [1, n]\backslash(\cup_{j=1}^{J}\mathcal{I}_j)$. Let $\mathbf{c}$ be a sequence satisfying $L(\mathbf{c}_{[1,n+k+1]}, \leq k) \leq T$. By virtue of Lemma 8.3.1, the bit c_i can be determined by

$$c_i = \boldsymbol{D}_{1, i-\sum_{j:\mathcal{I}_j\subseteq[1,i-1]}|\boldsymbol{\delta}_1\cap\mathcal{I}_j|} \tag{8.1}$$

for $i \in [1, n+k+1]\backslash(\cup_{j=1}^{J}\mathcal{I}_j)$. In addition, let $\mathcal{I}_j = [b_j^{min}, b_j^{max}]$ for $j \in [1, J]$ such that $b_j^{max} < b_{j-1}^{min}$ for $j \in [2, J]$. Again, since $\mathcal{I}_j$ is deletion isolated for $j \in [1, J]$, the submatrix

$$\boldsymbol{D}_{[1,d],[b_j^{min}-\sum_{i=j+1}^{J}|\boldsymbol{\delta_1}\cap\mathcal{I}_i|, b_j^{max}-\sum_{i=j}^{J}|\boldsymbol{\delta_1}\cap\mathcal{I}_i|]} \in \mathcal{D}_{|\boldsymbol{\delta_1}\cap[b_j^{min},b_j^{max}]|}(\mathbf{c}_{\mathcal{I}_j}, t_1, \dots, t_{d-1})$$

can be regarded as the d reads of the subsequence $\mathbf{c}_{\mathcal{I}_j}$ in a d head racetrack memory. According to Lemma 8.2.6, the bits $\mathbf{c}_{\mathcal{I}_j}$ with $|\boldsymbol{\delta}_1 \cap \mathcal{I}_j| < d$ can be recovered from

$$\boldsymbol{D}_{[1,d],[b_j^{min}-\sum_{i=j+1}^{J}|\boldsymbol{\delta_1}\cap\mathcal{I}_i|, b_j^{max}-\sum_{i=j}^{J}|\boldsymbol{\delta_1}\cap\mathcal{I}_i|]}$$

since the head distance satisfies $t_i \geq T[k(k-1)/2+1] + (7k-k^3)/6$ for $i \in \{1, \dots, d-1\}$. Note that there is at most a single interval $\mathcal{I}_{j_1}$ satisfying $|\boldsymbol{\delta}_1 \cap \mathcal{I}_{j_1}| \geq d$ when $k \leq 2d-1$. Hence, we are left to recover interval $\mathcal{I}_{j_1}$.

Split $\mathbf{c}_{[1,n+k+1]}$ into blocks

$$\mathbf{a}_i = \mathbf{c}_{[(i-1)B+1, \min\{iB, n+k+1\}]},\ i \in [1, \lceil(n+k+1)/B\rceil] \tag{8.2}$$

of length B except that $\mathbf{a}_{\lceil(n+k+1)/B\rceil}$ may have length shorter than B. Since $|\mathcal{I}_{j_1} \cap [1, n+k+1]| \leq B$, the interval $\mathcal{I}_{j_1}$ spans over at most two blocks $\mathbf{a}_{j_1'}$ and $\mathbf{a}_{j_1'+1}$. It then follows that there are at most two consecutive blocks, where $\mathcal{I}_{j_1}$ lies in, that remain to be recovered. Moreover, at most k deletions occur in interval $\mathcal{I}_{j_1}$, and hence in blocks $\mathbf{a}_{j'_1}$ and $\mathbf{a}_{j'+_1+1}$.

For an integer n and a sequence $\mathbf{c} \in \{0,1\}^{n+k+1}$ of length $n+k+1$, let the function S : $\{0,1\}^{n+k+1} \to \mathbb{F}_{4k\log B+o(\log B)}^{\lceil (n+k+1)/B\rceil}$ be defined by

$$S(\mathbf{c}) = (Hash(\mathbf{a}_1), Hash(\mathbf{a}_2), \ldots, Hash(\mathbf{a}_{\lceil (n+k+1)/B\rceil})), \tag{8.3}$$

where $\mathbf{a}_i$, $i \in [1, \lceil (n+k+1)/B\rceil]$ are the blocks of $\mathbf{c}$ defined in Eq. (8.2). The function $Hash(\mathbf{a}_{\lceil (n+k+1)/B\rceil})$, defined in Lemma 8.2.3, takes $\mathbf{a}_{\lceil (n+k+1)/B\rceil}$ as input of length at most B. The sequence $S(\mathbf{c})$ is a concatenation of the hashes $Hash$ of blocks of $\mathbf{c}$.

Lemma 8.3.2. *If $B > k$, there exists a function*

$$DecS : \{0,1\}^{n+1} \times \{0,1\}^{\lceil (n+k+1)/B\rceil (4\log B+o(\log B))} \to \{0,1\}^{n+k+1},$$

such that for any sequence $\mathbf{c} \in \{0,1\}^{n+k+1}$ and its length $n+1$ subsequence $\mathbf{d} \in \{0,1\}^{n+1}$, we have that $DecS(\mathbf{d}, S(\mathbf{c})) = \mathbf{c}$, i.e., the sequence $\mathbf{c}$ can be recovered from k deletions with the help of $S(\mathbf{c})$.

Proof. Note that $\mathbf{d}_{[(i-1)B+1, \min\{iB, n+k+1\}-k]}$ is a length $B-k$ subsequence of the block $\mathbf{a}_i$ for $i \in \{1, \ldots, \lceil (n+k+1)/B\rceil\}$. According to Lemma 8.2.3, the block $\mathbf{a}_i$ can be recovered from $\mathbf{d}_{(i-1)B+1, \max\{iB, n+k+1\}-k}$ with the help of $Hash(\mathbf{a}_i)$. Thus the sequence $\mathbf{c}$ can be recovered. □

We are now ready to present the code construction. For any sequence $\mathbf{c} \in \{0,1\}^n$, define the following encoding function:

$$Enc_1(\mathbf{c}) = (F(\mathbf{c}), R_1^{'}(\mathbf{c}), R_1^{''}(\mathbf{c})) \tag{8.4}$$

where

$$\begin{aligned} R_1^{'}(\mathbf{c}) &= ER(S(F(\mathbf{c}))), \\ R_1^{''}(\mathbf{c}) &= Rep_{k+1}(Hash(R_1'(\mathbf{c}))), \end{aligned} \tag{8.5}$$

and the function Rep_{k+1} is a $k+1$-fold repetition function that repeats each bit $k+1$ times. Note that we use $F(\mathbf{c}) \in \{0,1\}^{n+k+1}$ to obtain a sequence satisfying $L(F(\mathbf{c}), \leq k) \leq T$ so that Lemma 8.3.1 can be applied. The redundancy consists of two layers. The function $R_1^{'}(\mathbf{c})$ can be regarded as the first layer redundancy, with the help of which $F(\mathbf{c})$ can be recovered from k deletions. It computes the redundancy of a code that corrects two consecutive symbol erasures in $S_{k-m+1}(F(\mathbf{c}))$. The function $R_1^{''}(\mathbf{c})$ can be seen as the second layer redundancy that helps recover itself and $R_1^{'}(\mathbf{c})$ from k deletions.

When the head distance t_i satisfies $t_i = \max\{(3k + \lceil\log n\rceil + 2)[k(k-1)/2 + 1] + (7k - k^3)/6, (4k+1)(5k + \lceil\log n\rceil + 3)\}$ for $i \in [1, d-1]$, the length of $R_1^{'}(\mathbf{c})$ is given by $N_1 = 4\log B + o(\log B) = 4k\log\log n + o(\log\log n)$. The length of $R_1^{''}(\mathbf{c})$ is $N_2 = 8k^2(k+1)\log N_1 + O(1) = o(\log\log n)$. The length of the codeword $Enc_1(\mathbf{c})$ is given by $N = n + k + 1 + N_1 + N_2$. The next theorem proves Theorem 8.1.1 for cases when $d \leq k \leq 2d - 1$.

Theorem 8.3.1. *The set $C_1 = \{Enc_1(\mathbf{c}) : \mathbf{c} \in \{0,1\}^n\}$ is a d-head k-deletion correcting code for $d \leq k \leq 2d-1$, if the distance between any two consecutive heads satisfies $t_i = \max\{(3k + \lceil\log n\rceil + 2)[k(k-1)/2 + 1] + (7k - k^3)/6, (4k+1)(5k + \lceil\log n\rceil + 3)\}$, $i \in [1, d-1]$. The code C_1 can be constructed, encoded, and decoded in $O_k(poly(n))$ time. The redundancy of C_1 is $N - n = 4k\log\log n + o(\log\log n)$.*

Proof. For any $D \in D_k(\mathbf{c})$, let $\mathbf{d} = D_{1,[1,N-k]}$ be the first row of D, i.e., the first read. The sequence $\mathbf{d}$ is a length $N-k$ subsequence of $Enc_1(\mathbf{c})$. We first show how to recover $R_1^{'}(\mathbf{c})$ from $\mathbf{d}$. Note that $\mathbf{d}_{[N-N_2+1,N-k]}$ is a length $N_2 - k$ subsequence of $R_1^{''}(\mathbf{c})$, the $k+1$-fold repetition of $H(R_1^{'}(\mathbf{c}))$. Since a $k+1$-fold repetition code is a k deletion code, the hash function $Hash(R_1^{'}(\mathbf{c}))$ can be recovered. Furthermore, we have that $\mathbf{d}_{[n+k+2,n+k+1+N_1-k]}$ is a length $N_1 - k$ subsequence of $R_1^{'}(\mathbf{c})$. Hence according to Lemma 8.2.3, we can obtain $R_1^{'}(\mathbf{c})$ from $\mathbf{d}_{[n+k+2,n+k+1+N_1-k]}$, with the help of $Hash(R_1^{'}(\mathbf{c}))$.

Next, we show how to use $R_1^{'}(\mathbf{c})$ to recover $F(\mathbf{c})$. Note the fact that $L(F(\mathbf{c}), \leq k) \leq T$. From Lemma 8.3.1 and the discussion that follows, we can separate $F(\mathbf{c})$ into blocks $\mathbf{a}_i, i \in [1, \lceil(n+k+1)/B\rceil]$, of length B, and recover all but at most two consecutive blocks $\mathbf{a}_{j_1}$ and $\mathbf{a}_{j_1+1}$. This implies that $S(F(\mathbf{c}))$ can be retrieved with consecutive at most two symbol errors, the position of which can be identified. Hence we can use $R_1^{'}(\mathbf{c})$ to recover $S(F(\mathbf{c}))$ and find the hashes $Hash(\mathbf{a}_{j_1})$ and $Hash(\mathbf{a}_{j_1+1})$. Note that $D_{1,[1,n+1]}$ is a length $n+1$ subsequences of $F(\mathbf{c})$. Hence from Lemma 8.3.2 the sequence $F(\mathbf{c})$ and thus $\mathbf{c}$ can be recovered given $S(F(\mathbf{c}))$. The computation of $S(F(\mathbf{c}))$, which computes $O(n)$ times the hashes $Hash(\mathbf{a}_i)$, $[1, \lceil(n+k+1)/B\rceil]$, constitutes the main part of the computation complexity of $Enc_1(\mathbf{c})$. Since the computation of $Hash(\mathbf{a}_i)$ takes $O_k(\lceil B/\log n \rceil n \log^{2k} n) = O_k(n \log^{2k} n)$ time. It takes $O_k(poly(n))$ time to compute $Enc_1(\mathbf{c})$. □

8.4 Proof of Lemma 8.3.1

Let $\boldsymbol{D} \in \mathcal{D}_k(\mathbf{c}, t_1, \ldots, t_{d-1})$ be the d reads from all heads, where $\mathbf{c}$ satisfies

$$L(\mathbf{c}_{[1,n+k+1]}, \leq k) \leq T.$$

Then $\boldsymbol{D}$ is a d by $n+R-k$ matrix. The proof of Lemma 8.3.1 consists of two steps. The first step is to identify a set of disjoint intervals $\mathcal{I}_j^*$, $j \in [1, J]$ that satisfy:

(P1) There exists a set of disjoint and deletion isolated intervals $\mathcal{I}_j$, $j \in [1, J]$, such that $\boldsymbol{D}_{w,\mathcal{I}_j^*} = \mathbf{c}_{\mathcal{I}_j \cap \boldsymbol{\delta}_w^c}$ for $w \in [1, d]$ and $j \in [1, J]$, i.e., the subsequence $\boldsymbol{D}_{i,\mathcal{I}_j^*}$ comes from $\mathbf{c}_{\mathcal{I}_j}$ in the i-th read after deleting $\mathbf{c}_{\mathcal{I}_j \cap \boldsymbol{\delta}_i}$,

(P2) $J \leq k$ and $\boldsymbol{\delta}_w \subseteq \cup_{j=1}^J \mathcal{I}_j$ for $w \in [1, d]$,

(P3) $|\mathcal{I}_j^* \cap [1, n+1]| \leq B - k$.

The second step is to determine the number of deletions $|\boldsymbol{\delta}_i \cap \mathcal{I}_j|$ for $i \in [1, d]$ and $j \in [1, J]$, that happen in each interval in each head, based on $\boldsymbol{D}_{[1,d],\mathcal{I}_j^*}$. Then we have that

$$\mathcal{I}_j = [i_{2j-1} + \sum_{\ell=1}^{j-1} |\boldsymbol{\delta}_1 \cap \mathcal{I}_\ell|, i_{2j} + \sum_{\ell=1}^{j} |\boldsymbol{\delta}_1 \cap \mathcal{I}_\ell|],$$

where i_{2j-1} and i_{2j} are the starting and ending points of the interval $\mathcal{I}_j^*$. It is assumed that $i_j > i_l$ for $j > l$. The disjointness of $\mathcal{I}_j$, $j \in [1, J]$ follows from the fact that $\mathcal{I}_j^*$, $j \in [1, J]$ are disjoint. The two steps will be made explicit in the following two subsections respectively.

Identifying Intervals $\mathcal{I}_j^*$

The procedure for identifying intervals $\mathcal{I}_j^*$, $j \in [1, J]$, is as follows.

1. **Initialization:** Set all integers $m \in [1, n+R-k]$ unmarked. Let $i = 1$. Find the largest positive integer L such that the sequences $\boldsymbol{D}_{w,[i,i+L-1]}$ are equal for all $w \in [1, d]$. If such L exists and satisfies $L > t_{max}$, mark the integers $m \in [1, L - t_{max}]$ and go to Step 1. Otherwise, go to Step 1.

2. **Step 1:** Find the largest positive integer L such that the sequences $\boldsymbol{D}_{w,[i,i+L-1]}$ are equal for all $w \in [1,d]$. Go to Step 2. If no such L is found, set $L = 0$ and go to Step 2.

3. **Step 2:** If $L \geq 2t_{max} + T + 1$, mark the integers $m \in [i + t_{max}, \min\{i + L - 1, n+1\} - t_{max}]$. Set $i = i + L + 1$ and go to Step 3. Else $i = i + 1$ and go to Step 3.

4. **Step 3:** If $i \leq n + 1$, go to Step 1. Else go to Step 4.

5. **Step 4:** If the number of unmarked intervals[1] within $[1, n + 1]$ is not greater than k, output all unmarked intervals. Else output the first k intervals, i.e., the intervals with the minimum k starting indices.

We prove that the output intervals satisfy the above constraints. The following lemma will be used.

Lemma 8.4.1. *Let $\boldsymbol{D} \in \mathcal{D}_k(\mathbf{c})$ for some sequence $\mathbf{c}$ satisfying $L(\mathbf{c}_{[1,n+k+1]}, \leq k) \leq T$. Let $t_{max} = \max_{i\in[1,d-1]} t_i$ such that $t_w \geq k(T+1)+1$ for $w \in [1, d-1]$. If the sequences $\boldsymbol{D}_{w,[i_1,i_2]}$ are equal for all $w \in [1,d]$ in some interval $[i_1, i_2] \subseteq [1, n+1]$ with length $i_2 - i_1 + 1 \geq 2t_{max} + T + 1$, then no deletions occur within bits $\boldsymbol{D}_{w,[i_1+t_{max},i_2-t_{max}]}$ for all w, i.e., there exists integers $i'_1 = i_1 + t_{max} + |\boldsymbol{\delta}_j \cap [1, i'_1 - 1]|$ and $i'_2 = i_2 - t_{max} + |\boldsymbol{\delta}_j \cap [1, i'_2 - 1]|$, such that $\mathbf{c}_{[i'_1,i'_2]} = \boldsymbol{D}_{w,[i_1+t_{max},i_2-t_{max}]}$ and $[i'_1, i'_2] \cap \boldsymbol{\delta}_w = \emptyset$ for $w \in [1, d]$. In addition, both intervals $[1, i'_1 - 1]$ and $[i'_2 + 1, n + R]$ are deletion isolated.*

Proof. Let $c_{i'_0}$, $c_{i'_1}$, $c_{i'_2}$, and $c_{i'_3}$ be the bits that become $\boldsymbol{D}_{1,i_1}$, $\boldsymbol{D}_{1,i_1+t_{max}}$, $\boldsymbol{D}_{1,i_2-t_{max}}$, and $\boldsymbol{D}_{1,i_2}$ respectively after deletions, i.e., $i'_0 - |\boldsymbol{\delta}_1 \cap [1, i'_0 - 1]| = i_1$, $i'_1 - |\boldsymbol{\delta}_1 \cap [1, i'_1 - 1]| = i_1 + t_{max}$, $i'_2 - |\boldsymbol{\delta}_1 \cap [1, i'_2 - 1]| = i_2 - t_{max}$, and $i'_3 - |\boldsymbol{\delta}_1 \cap [1, i'_3 - 1]| = i_2$. We show that no deletions occur within $\boldsymbol{D}_{w,[i_1,i_2-t_{max}]}$ for $w \in [1, d-1]$ or within $\boldsymbol{D}_{w,[i_1+t_{max},i_2]}$ for $w \in [2, d]$, i.e., $\boldsymbol{\delta}_w \cap [i'_0, i'_2] = \emptyset$ for $w \in [1, d-1]$, and $\boldsymbol{\delta}_w \cap [i'_1, i'_3] = \emptyset$ for $w \in [2, d]$.

Suppose on the contrary, there are deletions within $\boldsymbol{D}_{w,[i_1,i_2-t_{max}]}$ for $w \in [1, d-1]$. Then there exist some $w_1 \in [1, d-1]$ and $k_1 \in [1, k]$, such that $\delta_{w_1,k_1} \in [i'_0, i'_2]$ (recall that δ_{w_1,k_1} is the location of the k_1-th deletion in the w_1-th read). Then we have that $\delta_{w_1+1,k_1} = \delta_{w_1,k_1} + t_{w_1} \in [i'_0, i'_3]$. Note that there are $k - k_1$

[1] An unmarked interval $[i, j]$ means that $m \in [i, j]$ are not marked and $i - 1$ and $j + 1$ are marked. It is assumed that 0 and $n + R - k + 1$ are marked.

deletions $\{\delta_{w_1,k_1+1},\ldots,\delta_{w_1,k}\}$ to the right of δ_{w_1,k_1} and k_1-1 deletions $\{\delta_{w_1+1,1},\ldots,\delta_{w_1+1,k_1-1}\}$ to the left of δ_{w_1+1,k_1}. Hence we have that

$$\begin{aligned}
&|(\boldsymbol{\delta}_{w_1}\cup\boldsymbol{\delta}_{w_1+1})\cap[\delta_{w_1,k_1}+1,\delta_{w_1,k_1}+t_{w_1}-1]|\\
\leq&|(\boldsymbol{\delta}_{w_1}\cup\boldsymbol{\delta}_{w_1+1})\cap[\delta_{w_1,k_1}+1,\delta_{w_1+1,k_1}-1]|\\
\leq&k-k_1+k_1-1\\
=&k-1,
\end{aligned}$$

meaning that there are at most $k-1$ deletions in the w_1-th or (w_1+1)-th heads that lie in interval $[\delta_{w_1,k_1}+1,\delta_{w_1,k_1}+t_{w_1}-1]$. Since $t_{w_1}\geq k(T+1)+1$, there are at least k disjoint intervals of length $T+1$ that lie in interval $[\delta_{w_1,k_1}+1,\delta_{w_1,k_1}+t_{w_1}-1]$. It then follows that there exists an interval $[i',i'+T]\subset[\delta_{w_1,k_1}+1,\delta_{w_1,k_1}+t_{w_1}-1]$ such that $[i',i'+T]\cap(\boldsymbol{\delta}_{w_1}\cup\boldsymbol{\delta}_{w_1+1})=\emptyset$. Let $l'_1=|\boldsymbol{\delta}_{w_1}\cap[1,i'-1]|$ and $l'_2=|\boldsymbol{\delta}_{w_1+1}\cap[1,i'-1]|$ be the number of deletions in heads w_1 and w_1+1 respectively that is to the left of i'. We have that $l'_1>l'_2$ since $\delta_{w_1,k_1}<i'$ and $\delta_{w_1+1,k_1}>i'+T$. Since $[i',i'+T]\cap(\boldsymbol{\delta}_{w_1}\cup\boldsymbol{\delta}_{w_1+1})=\emptyset$ and $l'_1-l'_2\leq k<T$, we have that

$$\begin{aligned}
l'_1=&|\boldsymbol{\delta}_{w_1}\cap[1,i'-1]|\\
=&|\boldsymbol{\delta}_{w_1}\cap[1,i'+l'_1-l'_2-1]|\\
=&|\boldsymbol{\delta}_{w_1}\cap[1,i'+T-1]|,\text{ and}\\
l'_2=&|\boldsymbol{\delta}_{w_1+1}\cap[1,i'-1]|\\
=&|\boldsymbol{\delta}_{w_1+1}\cap[1,i'+T+l'_2-l'_1-1]|.
\end{aligned}$$

Therefore,

$$\begin{aligned}
&\mathbf{c}_{[i'+l'_1-l'_2,i'+T]}\\
=&\boldsymbol{D}_{w_1,[i'+l'_1-l'_2-|\boldsymbol{\delta}_{w_1}\cap[1,i'+l'_1-l'_2-1]|,i'+T-|\boldsymbol{\delta}_{w_1}\cap[1,i'+T-1]|]}\\
=&\boldsymbol{D}_{w_1,[i'-l'_2,i'+T-l'_1]}\\
=&\boldsymbol{D}_{w_1+1,[i'-l'_2,i'+T-l'_1]}\\
=&\boldsymbol{D}_{w_1+1,[i'-|\boldsymbol{\delta}_{w_1+1}\cap[1,i'-1]|,i'+T+l'_2-l'_1-|\boldsymbol{\delta}_{w_1+1}\cap[1,i'+T+l'_2-l'_1-1]|]}\\
=&\mathbf{c}_{[i',i'+T+l'_2-l'_1]},
\end{aligned}$$

which implies that $L(\mathbf{c}_{[i',i'+T]},l'_1-l'_2)=T+1>T$. Since $[i',i'+T]\subset[i'_0,i'_3]\subset[1,n+k+1]$, this is a contradiction to the assumption that $L(\mathbf{c}_{[1,n+k+1]},l'_1-l'_2)\leq T$. Therefore, there are no deletions within $\boldsymbol{D}_{w,[i_1,i_2-t_{max}]}$ for $w\in[1,d-1]$, i.e., $\boldsymbol{\delta}_w\cap[i'_0,i'_2]=\emptyset$ for $w\in[1,d-1]$. Similarly, we have that $\boldsymbol{\delta}_w\cap[i'_1,i'_3]=\emptyset$

for $w \in [2, d]$. Since $[i'_1, i'_2] \subset [i'_0, i'_2]$ and $[i'_1, i'_2] \subset [i'_1, i'_3]$, it follows that

$$[i'_1, i'_2] \cap \boldsymbol{\delta}_w = \emptyset, \tag{8.6}$$

and hence $\mathbf{c}_{[i'_1,i'_2]} = \boldsymbol{D}_{w,[i_1+t_{max},i_2-t_{max}]}$ for $w \in [1, d]$.

Next we show that the intervals $[1, i'_1 - 1]$ and $[i'_2 + 1, n + R]$ are deletion isolated. Suppose on the contrary, there exists some $w_2 \in [1, d]$ for which $(\boldsymbol{\delta}_{w_2} \cap [1, i'_1 - 1]) + t_{w_2} \neq (\boldsymbol{\delta}_{w_2+1} \cap [1, i'_1 - 1])$. Then we have that $|\boldsymbol{\delta}_{w_2} \cap [1, i'_1 - 1]| > |\boldsymbol{\delta}_{w_2+1} \cap [1, i'_1 - 1]|$. Let $x = |\boldsymbol{\delta}_{w_2} \cap [1, i'_1 - 1]| - |\boldsymbol{\delta}_{w_2+1} \cap [1, i'_1 - 1]|$, then,

$$\begin{aligned}
&\mathbf{c}_{[i'_1,i'_2-x]}\\
\overset{(a)}{=}&\boldsymbol{D}_{w_2,[i_1+t_{max}+|\boldsymbol{\delta}_{w_2}\cap[1,i'_1-1]|,i_2-t_{max}-x+|\boldsymbol{\delta}_{w_2}\cap[1,i'_2-x-1]|]}\\
=&\boldsymbol{D}_{w_2+1,[i_1+t_{max}+|\boldsymbol{\delta}_{w_2}\cap[1,i'_1-1]|,i_2-t_{max}-x+|\boldsymbol{\delta}_{w_2}\cap[1,i'_2-x-1]|]}\\
=&\boldsymbol{D}_{w_2+1,[i_1+t_{max}+x+|\boldsymbol{\delta}_{w_2+1}\cap[1,i'_1-1]|,i_2-t_{max}+|\boldsymbol{\delta}_{w_2+1}\cap[1,i'_2-1]|]}\\
\overset{(b)}{=}&\mathbf{c}_{[i'_1+x,i'_2]},
\end{aligned} \tag{8.7}$$

where (a) and (b) hold since we have E.q. (8.6). This implies that

$$\begin{aligned}
L(\mathbf{c}_{i'_1,i'_2}, x) =& i'_2 - i'_1 + 1\\
\geq& T + 1,
\end{aligned}$$

which contradicts the fact that $L(\mathbf{c}_{[1,n+k-1],\leq k}) \leq T$. Therefore, the intervals $[1, i'_1 - 1]$ and $[i'_2 + 1, n + R]$ are deletion isolated. □

In the following, we show that the output intervals satisfy **(P1)**, **(P2)**, and **(P3)**, respectively. Let $[p_{2j-1}, p_{2j}]$, $j \in [1, J']$ be the marked intervals in the algorithm, where $p_1 < \ldots < p_{2J'}$. Let $p_0 = 0$ and $p_{2J'+1} = n + R + 1$, then the output intervals are the leftmost up to k nonempty intervals among $\{[p_{2j} + 1, p_{2j+1} - 1]\}_{j=0}^{J'}$. Note that from the marking operation in the **Initialization** step and **Step 2**, the interval $[n + 1 - t_{max}, n + R - k]$ is not marked. In addition, for any $j \in [1, J']$, sequences $\boldsymbol{D}_{w,[p_{2j-1},p_{2j}]}$ are equal for all $w \in [1, d]$. Hence, according to Lemma 8.4.1, there exist intervals $[p'_{2j-1}, p'_{2j}]$, $j \in [1, J']$, where

$$\begin{aligned}
&p'_j = p_j + |\boldsymbol{\delta}_w \cap [1, p'_j - 1]|, \text{ and}\\
&[p'_{2\ell-1}, p'_{2\ell}] \cap \boldsymbol{\delta}_w = \emptyset,
\end{aligned} \tag{8.8}$$

for all $j \in [1, 2J']$, $\ell \in [1, J']$, and $w \in [1, d]$. In addition, intervals $[1, p'_{2j-1} - 1]$ and $[p'_{2j}+1, n+R]$ are deletion isolated[2] for $j \in [1, J']$. It follows that $[p'_{2j-1}, p'_{2j+1} - 1]$ is deletion isolated for $j \in [1, J']$. Since $[p'_{2j-1}, p'_{2j}] \cap \boldsymbol{\delta}_w = \emptyset$ for $j \in [1, J']$ and $w \in [1, d]$, then we have that the intervals $[p'_{2j} + 1, p'_{2j+1} - 1]$, $j \in [0, J']$, where $p'_0 = 0$ and $p'_{2J+1} = n + R + 1$, are deletion isolated. From (8.8) we have that $\boldsymbol{D}_{w,[p_{2j}+1,p_{2j+1}-1]} = \mathbf{c}_{[p'_{2j}+1,p'_{2j+1}-1]\cap\delta^c_w}$. In addition, the intervals $\{[p'_{2j} + 1, p'_{2j+1} - 1]\}_{j=0}^{J'}$ are disjoint since

$$\begin{aligned}&(p'_{2(j+1)} + 1) - (p'_{2j+1} - 1)\\ =&p_{2(j+1)} + |\boldsymbol{\delta}_w \cap [1, p'_{2(j+1)} - 1]| + 2 - p_{2j+1} - |\boldsymbol{\delta}_w \cap [1, p'_{2j+1} - 1]|\\ \geq&T + |\boldsymbol{\delta}_w \cap [1, p'_{2(j+1)} - 1]| - |\boldsymbol{\delta}_w \cap [1, p'_{2j+1} - 1]|\\ \geq&T - k > 0,\end{aligned}$$

for $j \in [0, J' - 1]$. Therefore, the output intervals $\{[p_{2j} + 1, p_{2j+1} - 1]\}_{j=0}^{J'}$ satisfy **(P1)**.

Next, we show that the output intervals satisfy **(P2)**. For any output interval $[p_{2j} + 1, p_{2j+1} - 1]$ with $[p_{2j} + 1, p_{2j+1} - 1] \subseteq [1, n + 1 - t_{max}]$, the corresponding interval $[p'_{2j} + 1, p'_{2j+1} - 1]$ contains at least one deletion in $\boldsymbol{\delta}_w$, i.e., $[p'_{2j} + 1, p'_{2j+1} - 1] \cap \boldsymbol{\delta}_w \neq \emptyset$, for some $w \in [1, d]$. Otherwise, we have that $[p'_{2j}+1, p'_{2j+1}-1] \cap \boldsymbol{\delta}_w = \emptyset$ for $w \in [1, d]$. Combining with (8.8) and the fact that intervals $[1, p'_{2j-1} - 1]$ are deletion isolated for $j \in [1, J']$, it follows that the sequences $\boldsymbol{D}_{w,[p_{2j}+1,p_{2j+1}-1]}$ are equal for $w \in [1, d]$. This implies that the interval $[p_{2j} + 1, p_{2j+1} - 1]$ is marked during the procedure, which is a contradiction to the fact that $[p_{2j} + 1, p_{2j+1} - 1]$ is not marked. Therefore, there are at most k unmarked intervals that lie within the interval $[1, n + 1]$. Note that there is one unmarked interval $[n + 1 - t_{max}, n + R - k]$ the does not lie in $[1, n + 1]$. It follows that there are at most $k + 1$ unmarked intervals in total. When there are $k + 1$ unmarked intervals, the deletions $\boldsymbol{\delta}_w$ are contained in the k output intervals since each output interval within $[1, n + 1]$ contains at least one deletion. When there are no more than k intervals, the deletions are contained in the unmarked output intervals since the marked intervals do not contain deletions. Therefore we have that $\boldsymbol{\delta}_w \subseteq \{[p_{2j}+1, p_{2j+1}-1]\}_{j=1}^{J}$, where $\{[p_{2j}+1, p_{2j+1}-1]\}_{j=1}^{J}$ are the output intervals and $J \leq k$.

[2]The interval $[p_1, p_2]$ may be marked in the **Initialization** step and have length less than $T + 2t_{max} + 1$. In that case, apply Lemma 8.4.1 by considering an interval $[-t_{max} + T + 1, 0]$ where $\boldsymbol{D}_{w,[-t_{max}+T+1,0]}$ are equal for $w \in [1, d]$.

Finally, we show that $|\mathcal{I}_j^* \cap [1, n+1]| \le B - k$ for $j \in [1, J]$. We first prove that for any unmarked index $i \in [1, n+1 - \lfloor t_{max} + (T+1)/2 \rfloor]$, there exist some $w \in [1, d]$ and $k_1 \in [1, k]$, such that a deletion at δ_{w,k_1} occurs within distance $\lfloor t_{max} + (T+1)/2 \rfloor$ to the bit $\mathbf{c}_{i'=i+|\boldsymbol{\delta}_w \cap [1,i'-1]|}$ that becomes $\boldsymbol{D}_{w,i}$, i.e., $\delta_{w,k_1} \in [i' - \lfloor t_{max} + (T+1)/2 \rfloor, i' + \lfloor t_{max} + (T+1)/2 \rfloor]$[3]. Otherwise, we have that $[i' - \lfloor t_{max} + (T+1)/2 \rfloor, i' + \lfloor t_{max} + (T+1)/2 \rfloor] \cap \delta_w = \emptyset$ for $w \in [1, d]$. Since $[i' - \lfloor t_{max} + (T+1)/2 \rfloor, i' + \lfloor t_{max} + (T+1)/2 \rfloor]$ has length more than t_w for $w \in [1, d]$, we have that $\delta_{w+1,j} = \delta_{w,j} + t_w \in [1, i' - \lfloor t_{max} + (T+1)/2 \rfloor - 1]$ for every $\delta_{w,j} + t_w \in [1, i' - \lfloor t_{max} + (T+1)/2 \rfloor - 1]$. It follows that $[1, i' - \lfloor t_{max} + (T+1)/2 \rfloor - 1]$ is deletion isolated. Therefore, we have that

$$\begin{aligned}
&\boldsymbol{D}_{w,[i-\lfloor t_{max}+(T+1)/2\rfloor, i+\lfloor t_{max}+(T+1)/2\rfloor]} \\
=&\mathbf{c}_{[i-\lfloor t_{max}+(T+1)/2\rfloor+|\boldsymbol{\delta}_w \cap [i'-1]|, i+\lfloor t_{max}+(T+1)/2\rfloor+|\boldsymbol{\delta}_w \cap [i'-1]|]} \\
=&\mathbf{c}_{[i'-\lfloor t_{max}+(T+1)/2\rfloor, i'+\lfloor t_{max}+(T+1)/2\rfloor]}
\end{aligned}$$

are equal for all $w \in [1, d]$, which means that the interval $[i - \lfloor t_{max} + (T+1)/2 \rfloor, i + \lfloor t_{max} + (T+1)/2 \rfloor]$ and thus the index i should be marked. Therefore, every unmarked index $i \in [1, n+1 - \lfloor t_{max} + (T+1)/2 \rfloor]$ is associated with a deletion index δ_{w,k_1} that is within distance $\lfloor t_{max} + (T+1)/2 \rfloor$ to $i' = i + |\boldsymbol{\delta}_w \cap [1, i'-1]|$. On the other hand, any deletion δ_{w,k_1} is associated with at most $2\lfloor (2t_{max} + T + 1)/2 \rfloor + 1$ unmarked indices. Therefore, the number of unmarked bits within $[1, n+1 - \lfloor t_{max} + (T+1)/2 \rfloor]$ is at most $(2\lfloor (2t_{max} + T + 1)/2 \rfloor + 1)kd$. The number of unmarked bits within $[1, n+1]$ is at most $(2\lfloor (2t_{max} + T + 1)/2 \rfloor + 1)kd + \lfloor (2t_{max} + T + 1)/2 \rfloor = B - k$.

Determining the Number of Deletions

In this subsection we present the algorithm for determining the number of deletions $|\boldsymbol{\delta}_w \cap \mathcal{I}_j|$, $w \in [1, d]$, for any deletion isolated interval $\mathcal{I}_j \subseteq [1, n+k+1]$. The input for this algorithm is the reads $\boldsymbol{D}_{[1,d],\mathcal{I}_j^*}$ obtained by deleting $\mathbf{c}_{\boldsymbol{\delta}_w \cap \mathcal{I}_j}$, $w \in [1, d]$ from $\mathbf{c}_{\mathcal{I}_j}$. The interval $\mathcal{I}_j^*$ is the output interval obtained from the procedure in Subsection 8.4. Note that the interval $\mathcal{I}_j$ is not known at this point. In the algorithm only the first two reads $\boldsymbol{D}_{[1,2],\mathcal{I}_j^*}$ are used. Let $\mathcal{I}_j = [b_{min}, b_{max}]$ for some integers b_{min}

[3]When $i' - \lfloor t_{max} + (T+1)/2 \rfloor < 0$, consider bits $\boldsymbol{D}_{w,[i'-\lfloor t_{max}+(T+1)/2\rfloor,0]}$ that are equal for $w \in [1, d]$.

and b_{max}. Consider the following intervals,

$$\mathcal{B}_{i,m} = \begin{cases} [b_{min} + (i-1)t_1 + (m-1)(T+2k+1), \min\{b_{min} \\ +(i-1)t_1 + m(T+2k+1) - 1, b_{max}\}], \\ \text{for } i \in [1, \lceil (b_{max} - b_{min} + 1)/t_1 \rceil] \text{ and } m \in [1, \min\{4k+1, \\ \lceil ((b_{max} - b_{min} + 1) \bmod t_1)/(T+2k+1) \rceil\}] \end{cases}.$$

Recall that here t_1 is the distance between head 1 and head 2. The intervals $\mathcal{B}_{i,m}$ are disjoint and have length $T + 2k + 1$ except when $i = \lceil (b_{max} - b_{min} + 1)/t_1 \rceil$ and $m = \min\lceil ((b_{max} - b_{min} + 1) \bmod t_1)/(T + 2k + 1)\rceil$ the length might be less. Let $\mathcal{U}_m = \cup_i \mathcal{B}_{i,m}$ be the union of intervals $\mathcal{B}_{i,m}$ with the same m for $m \in [1, 4k+1]$. Then the unions U_m are disjoint since $t_1 \geq (4k + 1)(T + 2k + 1)$. Since the deletions occur in at most $2k$ positions in the first two heads, at least $2k + 1$ unions $\{\mathcal{U}_{m_1}, \ldots, \mathcal{U}_{m_{2k+1}}\}$ satisfy $\mathcal{U}_{m_l} \cap (\delta_1 \cup \delta_2) = \emptyset$ for $l \in [1, 2k + 1]$.

Similarly, let $\mathcal{I}_j = [b'_{min}, b'_{max}]$ for some integers b'_{min} and b'_{max}. Define the intervals

$$\mathcal{B}'_{i,m} = \begin{cases} [b'_{min} + (i-1)t_1 + (m-1)(T+2k+1), \min\{b'_{min} + (i-1)t_1 \\ +m(T+2k+1) - k - 1, b'_{max}\}], \\ \text{for } i \in [1, \lceil (b'_{max} - b'_{min} + 1)/t_1 \rceil] \text{ and } m \in [1, \min\{4k+1, \\ \lceil ((b'_{max} - b'_{min} + 1) \bmod t_1)/(T+2k+1) \rceil\}] \end{cases}.$$

Then $\mathcal{B}_{i,m}$ are disjoint length $T+k+1$ intervals except when $i = \lceil (b'_{max} - b'_{min} + 1)/t_1 \rceil$ and $m = \min\{4k + 1, \lceil ((b'_{max} - b'_{min} + 1) \bmod t_1)/(T + 2k + 1)\rceil\}$ the length might be less. Let

$$\mathcal{IM}' = \{(i, m) : |\mathcal{B}'_{i,m}| = T + k + 1\}$$

be the set of (i, m) pairs for which $\mathcal{B}'_{i,m}$ has length $T+k+1$. Since $|\mathcal{I}^*_j| = |\mathcal{I}_j| - |\mathcal{I}_j \cap \delta_w|$ for $w \in [1, d]$, we have that

$$\begin{aligned} b'_{max} - b'_{min} + 1 &= |\mathcal{I}^*_j| \\ &\leq |\mathcal{I}_j| = b_{max} - b_{min} + 1. \end{aligned}$$

It follows that $\mathcal{B}_{i,m} \neq \emptyset$ when $(i, m) \in \mathcal{IM}'$. For $(i, m) \in \mathcal{IM}'$, let $p_{i,m}$ and $q_{i,m}$ be the beginning and end points of interval $\mathcal{B}_{i,m}$, i.e., $\mathcal{B}_{i,m} = [p_{i,m}, q_{i,m}]$. Similarly, let $\mathcal{B}'_{i,m} = [p'_{i,m}, q'_{i,m}]$.

Since $\boldsymbol{D}_{w,\mathcal{I}_j^*}$ can be obtained by deleting bits $\mathbf{c}_{\delta_w \cap \mathcal{I}_j}$ from $\mathbf{c}_{\mathcal{I}_j}$, we have that

$$\begin{aligned}
&\boldsymbol{D}_{w,b'_{max}} = \mathbf{c}_{b'_{max}+x}\\
&\boldsymbol{D}_{w,b'_{min}+(i-1)t_1+(m-1)(T+2k+1)} = \mathbf{c}_{b_{min}+(m-1)t_1+(m-1)(T+2k+1)+y}, \text{ and}\\
&\boldsymbol{D}_{w,b'_{min}+(i-1)t_1+m(T+2k+1)-k-1} = \mathbf{c}_{b_{min}+(i-1)t_1+m(T+2k+1)-k-1+z}
\end{aligned}$$

for $w \in \{1,2\}$, $(i,m) \in \mathcal{IM}'$, and integers x, y, z that satisfy $b_{min} - b'_{min} \leq x, y, z \leq k$. Therefore, the sequence $\boldsymbol{D}_{w,\mathcal{B}'_{i,m}}$ is a length $T+k+1$ subsequence of $\mathbf{c}_{\mathcal{B}_{i,m}}$ for $w \in \{1,2\}$ and for all $i, m \in \mathcal{IM}'$.

The algorithm is given as follows.

1. **Step 1:** For all $(i,m) \in \mathcal{IM}'$, find a unique integer $0 \leq x_{i,m} \leq k$ such that $\boldsymbol{D}_{1,[p'_{i,m},q'_{i,m}-x_{i,m}]} = \boldsymbol{D}_{2,[p'_{i,m}+x_{i,m},q'_{i,m}]}$. If no or more than one such integers exist, let $x_{i,m} = 0$. Go to Step 2.

2. **Step 2:** For all $m \in [1, 4k+1]$, compute the sum $s_m = \sum_{i:(i,m)\in\mathcal{IJ}'} x_{i,m}$. Go to step 3.

3. **Step 3:** Output the majority among $\{s_m\}_{m=1}^{4k+1}$.

Note that the set $\mathcal{IM}'$ and the intervals $\mathcal{B}'_{i,m} = [p'_{i,m}, q'_{i,m}]$ can be determined from Subsection 8.4. We now show that the algorithm outputs $|\mathcal{I}_j \cap \boldsymbol{\delta}_1|$. It suffices to show that $s_{m_l} = |\mathcal{I}_j \cap \boldsymbol{\delta}_1|$ for $l \in [1, 2k+1]$. First, we show that the unique integer x_{i,m_l} satisfying $\boldsymbol{D}_{1,[p'_{i,m_l},q'_{i,m_l}-x_{i,m_l}]} = \boldsymbol{D}_{2,[p'_{i,m_l}+x_{i,m_l},q'_{i,m_l}]}$ exists for $l \in [1, 2k+1]$ and i such that $(i, m_l) \in \mathcal{IM}'$. Moreover, the integer x_{i,m_l} equals $|\boldsymbol{\delta}_1 \cap [p_{1,1}, p_{i,m_l} - 1]| - |\boldsymbol{\delta}_2 \cap [p_{1,1}, p_{i,m_l} - 1]|$, the difference between the number of deletions in the first two heads that happen before the interval $\mathcal{B}_{i,m_l}$. Recall that m_l satisfies $\mathcal{U}_{m_l} \cap \boldsymbol{\delta}_w = \emptyset$ for $w \in \{1,2\}$ and that $\boldsymbol{D}_{w,[p'_{i,m_l},q'_{i,m_l}]}$ is a subsequence of $\mathbf{c}_{[p_{i,m_l},q_{i,m_l}]}$. Hence, let $x = |\boldsymbol{\delta}_1 \cap [p_{1,1}, p_{i,m_l} - 1]| - |\boldsymbol{\delta}_2 \cap [p_{1,1}, p_{i,m_l} - 1]|$, we have that

$$\begin{aligned}
&\boldsymbol{D}_{1,[p'_{i,m_l},q'_{i,m_l}-x]}\\
=&\mathbf{c}_{[p'_{i,m_l}+|\boldsymbol{\delta}_1\cap[p_{1,1},p_{i,m_l}-1]|,q'_{i,m_l}+|\boldsymbol{\delta}_1\cap[p_{1,1},p_{i,m_l}-1]|-x]}\\
=&\boldsymbol{D}_{2,[p'_{i,m_l}+x,q'_{i,m_l}]}.
\end{aligned} \tag{8.9}$$

Therefore, the integer $x_{i,m_l} = x$ satisfies $\boldsymbol{D}_{1,[p'_{i,m_l},q'_{i,m_l}-x_{i,m_l}]} = \boldsymbol{D}_{2,[p'_{i,m_l}+x_{i,m},q'_{i,m_l}]}$. We show this x_{i,m_l} is unique. Suppose there exists another integer $y > x$ for which $\boldsymbol{D}_{1,[p'_{i,m_l},q'_{i,m_l}-y]} = \boldsymbol{D}_{2,[p'_{i,m_l}+y,q'_{i,m_l}]}$. Then we have that

$$\boldsymbol{D}_{1,[p'_{i,m_l},q'_{i,m_l}-y]}$$

$$
\begin{aligned}
&= \boldsymbol{D}_{2,[p'_{i,m_l}+y, q'_{i,m_l}]} \\
&\overset{(a)}{=} \boldsymbol{D}_{1,[p'_{i,m_l}+y-x, q'_{i,m_l}-x]} \\
&= \mathbf{c}_{[p'_{i,m_l}+|\boldsymbol{\delta}_1 \cap [p_{1,1}, p_{i,m_l}-1]|+y-x, q'_{i,m_l}+|\boldsymbol{\delta}_1 \cap [p_{1,1}, p_{i,m_l}-1]|-x]},
\end{aligned}
$$

where (a) follows from Eq. (8.9). Since,

$$
\boldsymbol{D}_{1,[p'_{i,m_l}, q'_{i,m_l}-y]} = \mathbf{c}_{[p'_{i,m_l}+|\boldsymbol{\delta}_1 \cap [p_{1,1}, p_{i,m_l}-1]|, q'_{i,m_l}+|\boldsymbol{\delta}_1 \cap [p_{1,1}, p_{i,m_l}-1]|-y]},
$$

it follows that

$$
\begin{aligned}
&\mathbf{c}_{[p'_{i,m_l}+|\boldsymbol{\delta}_1 \cap [p_{1,1}, p_{i,m_l}-1]|, q'_{i,m_l}+|\boldsymbol{\delta}_1 \cap [p_{1,1}, p_{i,m_l}-1]|-y]} \\
=&\mathbf{c}_{[p'_{i,m_l}+|\boldsymbol{\delta}_1 \cap [p_{1,1}, p_{i,m_l}-1]|+y-x, q'_{i,m_l}+|\boldsymbol{\delta}_1 \cap [p_{1,1}, p_{i,m_l}-1]|-x]}.
\end{aligned}
$$

It then follows that

$$
\begin{aligned}
&L(\mathbf{c}_{[p'_{i,m_l}+|\boldsymbol{\delta}_1 \cap [p_{1,1}, p_{i,m_l}-1]|, q'_{i,m_l}+|\boldsymbol{\delta}_1 \cap [p_{1,1}, p_{i,m_l}-1]|-x]}, y-x) \\
=&q'_{i,m_l} - x - p'_{i,m_l} + 1 \\
\geq&T + k + 1 - k + 1 \\
\geq&T + 1,
\end{aligned}
$$

which is a contradiction to the fact that $L(\mathbf{c}, \leq k) \leq T$. Similarly, such contradiction occurs when $y < x$. Hence such x_{i,m_l} is unique.

Next, we show that $s_{m_l} = |\boldsymbol{\delta}_1 \cap \mathcal{I}_j|$ for $l \in [1, 2k+1]$. Since $p_{i,m_l} - p_{i-1,m_l} = t_1$ for $i \in [2, \max_{(i,m)\in\mathcal{I}\mathcal{M}'} i]$, we have that

$$
\begin{aligned}
&|\boldsymbol{\delta}_1 \cap [p_{1,1}, p_{i,m_l}-1]| \\
=&|\boldsymbol{\delta}_1 \cap [p_{1,1}, p_{1,m_l}-1]| + \sum_{w=1}^{i-1} |\boldsymbol{\delta}_1 \cap [p_{w,m_l}, p_{w+1,m_l}-1]| \\
\overset{(a)}{=}&|\boldsymbol{\delta}_2 \cap [p_{2,1}, p_{2,m_l}-1]| + \sum_{w=1}^{i-2} |\boldsymbol{\delta}_2 \cap [p_{w+1,m_l}, p_{w+2,m_l}-1]| \\
&+ |\boldsymbol{\delta}_1 \cap [p_{i-1,m_l}, p_{i,m_l}-1]| \\
=&|\boldsymbol{\delta}_2 \cap [p_{1,1}, p_{i,m_l}-1]| + |\boldsymbol{\delta}_1 \cap [p_{i-1,m_l}, p_{i,m_l}-1]|,
\end{aligned}
$$

where (a) hold since $|\boldsymbol{\delta}_1 \cap [p_{w-1,m_l}, p_{w,m_l}-1]| = |\boldsymbol{\delta}_2 \cap [p_{w,m_l}, p_{w+1,m_l}-1]|$ for $w \in [2, i-1]$. It then follows that $x_{i,m_l} = |\boldsymbol{\delta}_1 \cap [p_{i-1,m_l}, p_{i,m_l}-1]|$ ($p_{0,m_l} = p_{1,1}$) and that

$$
s_{m_l} = |\boldsymbol{\delta}_1 \cap [p_{1,1}, p_{\max_{(i,m)\in\mathcal{I}\mathcal{M}'} i, m_l} - 1]|.
$$

Note that $\boldsymbol{\delta}_1 \cap [p_{\max_{(i,m)\in\mathcal{I}\mathcal{M}'} i, m_l}, b_{max}] \subseteq \boldsymbol{\delta}_1 \cap [b_{max} - t_1 + 1, b_{max}]$. Since $\boldsymbol{\delta}_1 \cap [b_{max} - t_1 + 1, b_{max}] = \emptyset$, we have that $s_{m_l} = |\boldsymbol{\delta}_1 \cap \mathcal{I}_j|$. Then the majority rule works.

8.5 Correcting $k \geq 2d$ Deletions

In this section we present the code for correcting $k \geq 2d$ deletions as well as a lower bound on the redundancy when $t_i = o(n)$. The code construction is similar to the one presented in Sec. 8.3. We use Lemma 8.3.1 to identify the location of deletions within a set of disjoint intervals $\mathcal{I}_j$, each with length no more than B. Note that in order to apply Lemma 8.3.1, the sequence $\mathbf{c} \in \{0,1\}^n$ has to be transformed into a sequence $F(\mathbf{c}) \in \{0,1\}^{n+k+1}$ (recall Lemma 8.2.5) that satisfies $L(F(\mathbf{c}), \leq k) \leq T$. Then we use a concatenated code construction. Specifically, to protect a sequence $\mathbf{c} \in \{0,1\}^{n+k+1}$ from k deletions, we split $\mathbf{c}$ into blocks $\mathbf{a}_i$, $i \in [1, \lceil (n+k+1)/B \rceil]$ of length B as in Eq. (8.2). Then the function S defined in Eq. (8.3), which is a concatenation of hashes $Hash$ (see Lemma 8.2.3) of $\mathbf{a}_i$, $i \in [1, \lceil (n+k+1)/B \rceil]$, can be used to corret k deletions in $\mathbf{c}$ (see Lemma 8.3.2). Finally, a Reed-Solomon code is used to protect the S hashes. The encoding function is as follows

$$Enc_2(\mathbf{c}) = (F(\mathbf{c}), R_2^{'}(\mathbf{c}), R_2^{''}(\mathbf{c})) \tag{8.10}$$

where

$$\begin{aligned} R_2^{'}(\mathbf{c}) &= RS_{2\lfloor k/d \rfloor}(S(F(\mathbf{c}))), \\ R_2^{''}(\mathbf{c}) &= Rep_{k+1}(Hash(R_2^{'}(\mathbf{c}))), \end{aligned} \tag{8.11}$$

and function $S(\cdot)$ is defined in (8.3). The length of $R_2^{'}(\mathbf{c})$ is $N_1 = 2\lfloor k/d \rfloor \max\{\log(n+k+1), (4k \log B + o(\log B))\} = 2\lfloor k/d \rfloor \log n + o(\log n)$. The length of $R_2^{''}(\mathbf{c})$ is $N_2 = 8k^2(k+1)\log N_1 + O(1) = o(\log n)$. The length of $Enc_2(\mathbf{c})$ is $N = n + k + 1 + N_1 + N_2 = n + 2\lfloor k/d \rfloor \log n + o(\log n)$.

Theorem 8.5.1. *The set $C_2 = \{Enc_2(\mathbf{c}) : \mathbf{c} \in \{0,1\}^n\}$ is a d-head k-deletion correcting code for $2d \leq k$, if the distance between any two consecutive heads satisfies $t_i = T_{min}$, $i \in [1, \ldots, d-1]$. The code C_2 can be constructed, encoded, and decoded in n^{2k+1} time. The redundancy of C_2 is $N - n = +2\lfloor k/d \rfloor \log n + o(\log n)$.*

Proof. The proof is essentially the same to the proof of Theorem 8.3.1. For any $D \in \mathcal{D}_k(\mathbf{c})$, let $\mathbf{d} = \boldsymbol{D}_{1,[1,N-k]}$ be the first row of D. The sequence $\mathbf{d}$ is a length $N-k$ subsequence of $Enc_2(\mathbf{c})$. Then it is possible to recover $Hash(R_2^{'}(\mathbf{c}))$ from $R_2^{''}(\mathbf{c})$ and further recover $R_2^{'}(\mathbf{c})$.

It suffices to show how to use $R'(\mathbf{c})$ to recover $F(\mathbf{c})$. According to Lemma 8.3.1, we can identify a set of $J \leq k$ intervals $\{\mathcal{I}_j\}_{j=1}^J$, each with length not greater than B, such that $\boldsymbol{\delta}_1 \subseteq (\cup_{j=1}^J \mathcal{I}_j)$. Note that according to Lemma 8.2.6, the bits $\mathbf{c}_{\mathcal{I}_j}$

with$|\boldsymbol{\delta}_w| \cap \mathcal{I}_j| \leq d-1$ errors can be recovered, when $t_i \geq \max\{(3k+\lceil\log n\rceil+2)[k(k-1)/2+1]+(7k-k^3)/6, (4k+1)(5k+\lceil\log n\rceil+3)\}$. Note that each interval $\mathcal{I}_j$ with $|\boldsymbol{\delta}_w| \cap \mathcal{I}_j| \geq d$ spans over at most two blocks $\mathbf{a}_i$. Therefore, at most $2\lfloor k/d\rfloor$ blocks, the indices of which can be identified, contain at least d deletions. Hence the sequence $S(F(\mathbf{c}))$ can be recovered with at most $2\lfloor k/d\rfloor$ symbol errors, with known error locations. With the help of the Reed-Solomon code redundancy $RS_{2\lfloor k/d\rfloor}(S(F(\mathbf{c})))$, the sequence $S(F(\mathbf{c}))$ can be recovered. Then from Lemma 8.3.2 and Lemma 8.2.5 the sequence $F(\mathbf{c})$ and thus $\mathbf{c}$ can be recovered. The computation complexity of $Enc_2(\mathbf{c})$ has the same order as that of $Enc_1(\mathbf{c})$. It takes $O_k(poly(n))$ time to construct , encode, and decode $Enc_2(\mathbf{c})$. □

Now we present a lower bound on the redundancy for small head distances $t_i = o(n)$, $i \in [1, d-1]$, which proves the last part of Theorem 8.1.1.

Theorem 8.5.2. *Let C be a d-head k-deletion code with length n. If the distance t_i satisfies $t_i = n^{o(1)}$ for $i \in [1, d-1]$, then we have that $|C| \leq 2^{1/2\lfloor k/d\rfloor \log n + o(\log n)}$.*

Proof. Let $T_{sum} = \sum_{i=1}^{d-1} t_i$. Sample the sequence $\mathbf{c}$ with period T_{sum},

$$\mathbf{c}' = (c_{1+T_{sum}}, c_{1+3T_{sum}}, \ldots, c_{1+(2j+1)T_{sum}}, \ldots, c_{1+(2\lfloor(n-1-T_{sum})/2T_{sum}\rfloor-1)T_{sum}}).$$

We show that correcting k deletions in $\mathbf{c}$ is at least as hard as correcting $\lfloor k/d\rfloor$ erasures in $\mathbf{c}'$. It suffices to show that d deletions in heads $i \in [1, d]$ can erase the information of any bit in $\mathbf{c}'$. For $j \in [1, \lfloor(n-1-T_{sum})/2T_{sum}\rfloor]$, let the d deletions occur at positions

$$\{1+(2j-1)T_{sum} - \sum_{i=1}^{w} t_i : w \in [0, d-1]\}$$

at head 1. Then the corresponding d deletion in head m occur at positions

$$\{1+(2j-1)T_{sum} - \sum_{i=1}^{w} t_i + \sum_{i=1}^{m} t_i : w \in [0, d-1]\}.$$

It follows that the bit $c_{1+(2j-1)T_{sum}}$ is deleted in all heads. Suppose a genie tells the locations and values of all the d deleted bits in each head except the value of the bit $c_{1+(2j-1)T_{sum}}$. Then this reduces to a erasure of the bit $c_{1+(2j-1)T_{sum}}$ in $\mathbf{c}'$. Note that in this way, k deletions in $\mathbf{c}$ can cause $\lfloor k/d\rfloor$ erasures in $\mathbf{c}'$. From the Hamming bound, the size $|C|$ is upper bounded by

$$|C| \leq 2^n/(\sum_{i=1}^{\lfloor k/2d\rfloor} \binom{\lfloor(n-1-T_{sum})/2T_{sum}\rfloor}{i})$$

$$=2^{n-\lfloor k/2d\rfloor(\log n-\log(2T_{sum}))+o(\log n)}$$
$$=2^{n-\lfloor k/2d\rfloor\log n+o(\log n)}.$$

According to Theorem 8.5.2, the redundancy of a d head k deletion code is lower bounded by $\lfloor k/2d\rfloor\log n+o(\log n)$. □

8.6 Correcting k Deletions and Insertions

In this section we show how to correct a combination of up to k deletions and insertions in the d-head racetrack memory. In this scenario, more challenges arise since there may not be "shifts" between different reads, as we observed in Lemma 8.4.1, after a combination of deletions and insertions. This makes detection of errors harder. Moreover, Lemma 8.2.6 does not apply.

The encoding and decoding algorithms for this task can be regarded as a generalization of the algorithms for correcting k deletions. Similar to the idea in Sec. 8.3 and Sec. 8.5, we notice that the location of errors $(\boldsymbol{\delta}_i,\boldsymbol{\gamma}_i)$, $i\in[1,d]$ are contained in a set of disjoint edit isolated intervals, each with length at most B_E-k. Yet, different from the cases in Sec. 8.3 and Sec. 8.5, some of the edit isolated intervals cannot be detected and identified from the reads. Fortunately, the intervals that cannot be detected contain at least $2d$ errors in each read. In addition, the "shift" in bits outside the edit isolated intervals, caused by the errors in those edit isolated intervals, can be determined in a similar manner to the one in Sec. 8.4. Therefore, the bits outside the edit isolated intervals can be recovered similarly as in Sec. 8.3 and Sec. 8.5. In addition, we will prove a similar result to Lemma 8.2.6, for correcting both deletions and insertions, similar to what we did for correcting deletion errors. Specifically, we will show that the intervals with less than d errors can be recovered using the reads. Then, by using Reed-Solomon code to protect the deletion correcting hashes as we did in Sec. 8.3 and Sec. 8.5, the $2\lfloor k/d\rfloor\log n+o(\log n)$ redundancy can be achieved. We note that in this section, we let the head distances $t_i=t$ to be equal for $i\in[1,d-1]$.

We begin with the the algorithm for identifying a set of intervals $[b_{1j},b_{2j}]$, $j\in[1,J]$, such that for each $j\in[1,J]$, there is an interval $[p_{1j},p_{2j}]$ satisfying:

((A) $[p_{1j},p_{2j}]\subseteq[b_{1j},b_{2j}]$,

(B) $\boldsymbol{E}_{w,i}=\boldsymbol{E}_{w',i}$ for any $w,w'\in[1,d]$ and $i\in([b_{1j},p_{1j}-1]\cup[p_{2j}+1,b_{2j}])$,

(C) $\boldsymbol{E}_{[1,d],[p_{1j},p_{2j}]}\in\mathcal{E}_{k'}(\mathbf{c}_{\mathcal{I}_j})$ for some edit isolated interval $\mathcal{I}_j$ and $k'\geq 1$,

(D) $|[b_{1j}, b_{2j}]| \leq (2kdt + 2t + 1)(k + 1) + kdt + 2k$ for $j \in [1, J]$.

The algorithm is similar to the one in Sec. 8.4. However, different from the intervals $\mathcal{I}_j^*$, $j \in [1, J]$ generated in Sec. 8.4, which satisfy properties **(P1)** and **(P2)** in Sec. 8.4, here we do not necessarily have $\boldsymbol{E}_{[1,d],[b_{1j},b_{2j}]} \in \mathcal{E}_{k'}(\mathbf{c}_{\mathcal{I}_j'})$ for some edit isolated interval $\mathcal{I}_j'$, $j \in [1, J]$. Also, the error locations $(\gamma_w \cup \delta_w)$, $w \in [1, d]$ may not be contained in the collection of intervals $\cup_{j=1}^J \mathcal{I}_j$. Given a read matrix $\boldsymbol{E} \in \mathcal{E}_k(\mathbf{c})$, where $\mathbf{c} \in \{0, 1\}^{n+k+1}$ is a binary input. The algorithm is given as follows.

1. **Initialization:** Set all integers $m \in [1, n']$ unmarked, where n' is the number of columns in $\boldsymbol{E}$. Let $i = 1$. Find the largest positive integer L such that the sequences $\boldsymbol{E}_{w,[i,i+L-1]} = \boldsymbol{E}_{w',[i,i+L-1]}$ for any $w, w' \in [1, d]$. If such L exists and satisfies $L > kdt + t$, mark the integers $m \in [1, L - (kdt + t)]$ and go to Step 1. Otherwise, go to Step 1.

2. **Step 1:** Find the largest positive integer L such that the sequences $\boldsymbol{E}_{w,[i,i+L-1]} = \boldsymbol{E}_{w',[i,i+L-1]}$ for any $w, w' \in [1, d]$. Go to Step 2. If no such L is found, set $L = 0$ and go to Step 2.

3. **Step 2:** If $L \geq 2(kdt + t) + 1$, mark the integers $m \in [i + kdt + t, \min\{i + L - 1, n'\} - (kdt + t)]$. Set $i = i + L + 1$ and go to Step 3. Else $i = i + 1$ and go to Step 3.

4. **Step 3:** If $i \leq n'$, go to Step 1. Else go to Step 4.

5. **Step 4:** Output all unmarked intervals.

We now show that the output intervals satisfy the properties **(A)**, **(B)**, **(C)**, and **(D)** above.

Lemma 8.6.1. *For a read matrix $\boldsymbol{E} \in \mathcal{E}_k(\mathbf{c}) \in \{0, 1\}^{d \times n'}$, Let $[b_{1j}, b_{2j}]$, $j \in [1, J]$ be the output intervals in the above procedure such that $b_{11} < b_{12} < \ldots < b_{1J}$. There exists a set of intervals $[p_{1j}, p_{2j}]$, $j \in [1, J]$, satisfying* ***(A)**, **(B)**, **(C)**, and **(D)*** *above.*

Proof. Note that for each interval $[b_{1j}, b_{2j}]$, we have $\boldsymbol{E}_{w,[b_{1j},b_{1j}+kdt+t-1]} = \boldsymbol{E}_{w',[b_{1j},b_{1j}+kdt+t-1]}$ and $\boldsymbol{E}_{w,[b_{2j}-kdt-t+1,b_{2j}]} = \boldsymbol{E}_{w',[b_{2j}-kdt-t+1,b_{2j}]}$ for any $w, w' \in [1, d]$, except for $j = 1$, it is possible that b_{1j} can be less than 1, in which case, we can assume $\boldsymbol{E}_{w,i} = \boldsymbol{E}_{w',i}$ for any $i \leq 0$ and $w, w' \in [1, d]$. Consider the set of intervals $[b_{1j}+(i-1)t, b_{1j}+it-1]$ for $i \in [1, kd+1]$. Note that an error occurs in at most d intervals in one of the d heads. Therefore, at most kd intervals contain errors. Then, there exists an interval $[b_{1j}+(i_1-1)t, b_{1j}+i_1t-1]$ for some $i_1 \in [1, kd+1]$ such that $[b_{1j}+(i_1-1)t, b_{1j}+i_1t-1] \cap (\gamma_w \cup \boldsymbol{\delta}_w) = \emptyset$ for $w \in [1,d]$. Similarly, there exists an interval $[b_{2j}-i_2t+1, b_{2j}-(i_2-1)t]$ for some $i_2 \in [1, kd+1]$, such that $[b_{2j}-i_2t+1, b_{2j}-(i_2-1)t] \cap (\gamma_w \cup \boldsymbol{\delta}_w) = \emptyset$ for $w \in [1,d]$. This implies that $[b_{1j}+i_1t-1-k, b_{2j}-i_2t+1+k]$ is an edit isolated interval. Let $\boldsymbol{E}_{[1,d],[p_{1j},p_{2j}]} \in \mathcal{E}_{k'_j}(\mathbf{c}_{[b_{1j}+i_1t-1-k, b_{2j}-i_2t+1+k]})$, where $k'_j = |[b_{1j}+i_1t-1-k, b_{2j}-i_2t+1+k] \cap \boldsymbol{\delta}_1| + |[b_{1j}+i_1t-1-k, b_{2j}-i_2t+1+k] \cap \gamma_1|$, be the read matrix obtained from $\mathbf{c}_{[b_{1j}+i_1t-1-k, b_{2j}-i_2t+1+k]}$ after deletion errors at locations $\boldsymbol{\delta}_w \cap [b_{1j}+i_1t-1-k, b_{2j}-i_2t+1+k]$ and insertion errors at locations $\gamma_w \cap [b_{1j}+i_1t-1-k, b_{2j}-i_2t+1+k]$, $w \in [1,d]$. Then we have that $p_{1j} \in [b_{1j}+t-1-2k, b_{1j}+kdt+t-1]$ and $p_{2j} \in [b_{2j}-kdt-t+1, b_{2j}-t+1+2k]$. Therefore, the intervals $[p_{1j}, p_{2j}]$, $j \in [1,J]$ satisfy **(A)**, **(B)**. To show that $[p_{1j}, p_{2j}]$, $j \in [1,J]$ satisfy **(C)**, we need to show $k'_j \geq 1$ for each j. Suppose on the contrary, $k'_j = 0$. Then we since $\boldsymbol{E}_{[1,d],[p_{1j},p_{2j}]} \in \mathcal{E}_{k'_j}(\mathbf{c}_{[b_{1j}+i_1t-1-k, b_{2j}-i_2t+1+k]})$, we have that $\boldsymbol{E}_{w,[p_{1j},p_{2j}]} = \boldsymbol{E}_{w',[p_{1j},p_{2j}]}$ for any $w, w' \in [1,d]$. Then we have $\boldsymbol{E}_{w,[b_{1j},b_{2j}]} = \boldsymbol{E}_{w',[b_{1j},b_{2j}]}$ for any $w, w' \in [1,d]$, and $b_{1j}+kdt+t$ should have been marked, a contradiction.

Finally, we show that $|[b_{1j}, b_{2j}]| < (2kdt+2t+1)(k+1)+kdt+2k$. Note that an error that occurs at location i in the first head occurs at $i+(w-1)t$ in the w-th head. These locations are contained in an interval $[i, i+(d-1)t]$ of length less than dt. The locations of k errors in d heads are contained in k intervals, each of length at most dt. If $|[b_{1j}, b_{2j}]| \geq (2kdt+2t+1)(k+1)+kdt+2k$, there exists a sub-interval $[b'_{1j}, b'_{2j}] \subseteq [b_{1j}+k, b_{2j}-k]$ with length at least $2kdt+2t+1$, that is disjoint with the k intervals that contain locations of all errors in all heads. Since the interval $[b'_{1j}, b'_{2j}]$ has length more than t, the intervals $[1, b'_{1j}-1]$ and $[b'_{2j}+1, n+k+1]$ are edit disjoint, where $n+k+1$ is the length of $\mathbf{c}$. Moreover, $\boldsymbol{E}_{w,i} = \boldsymbol{E}_{w',i}$ for any $w, w' \in [1,d]$ and $i \in [b'_{1j} - |\boldsymbol{\delta}_1 \cap [1, b'_{1j}-1]| + |\gamma_1 \cap [1, b'_{1j}-1]|, b'_{2j} - -|\boldsymbol{\delta}_1 \cap [1, b'_{1j}-1]| + |\gamma_1 \cap [1, b'_{1j}-1]|]$. This implies that $i = b'_{1j} - |\boldsymbol{\delta}_1 \cap [1, b'_{1j}-1]| + |\gamma_1 \cap [1, b'_{1j}-1] + kdt + t$ should be marked, contradicting to the fact that $[b'_{1j}, b'_{2j}]$ is unmarked, $j \in [1,J]$. Therefore, we proved **(D)**. □

In the remainder of this section, we first show how to correct $k < d$ deletions and insertions in total. Then, we show how to determine the shifts caused by errors in the isolated intervals, and show that for those isolated intervals that can not be detected, there are at least $2d$ errors, and no shifts caused by these isolated intervals. Finally, we present our encoding and decoding algorithms for the general cases when $k \geq d$. The code is the same as the construction in Sec. 8.5, but with a different decoding algorithm. Before dealing with the $k < d$ case, we present a proposition that is repeatedly used in this section.

Proposition 8.6.1. *Let $\boldsymbol{E} \in \mathcal{E}_k(\mathbf{c})$ be a read matrix for some sequence $\mathbf{c}$ satisfying $L(\mathbf{c}, \leq k) \leq T$. For any integers $i \in [1, n]$ and $w, w' \in [1, d]$ such that no error occurs in interval $[i - T - 2k, i]$ in the w-th and w'-th head, i.e.,*

$$\begin{aligned}&(\boldsymbol{\delta}_w \cup \gamma_w) \cap [i - T - 2k, i] = \emptyset, \text{ and}\\ &(\boldsymbol{\delta}_{w'} \cup \gamma_{w'}) \cap [i - T - 2k, i] = \emptyset,\end{aligned} \tag{8.12}$$

If

$$\boldsymbol{E}_{w,[i-T-2k,i-x]} = \boldsymbol{E}_{w',[i-T-2k+x,i]} \tag{8.13}$$

for some integer $x \in [0, k]$, then

$$\begin{aligned}&|\gamma_w \cap [1, i - T - 2k - 1]| - |\boldsymbol{\delta}_w \cap [1, i - T - 2k - 1]| + x\\ =&|\gamma_{w'} \cap [1, i - T - 2k - 1]| - |\boldsymbol{\delta}_{w'} \cap [1, i - T - 2k - 1]|.\end{aligned} \tag{8.14}$$

Proof. Suppose on the contrary,

$$\begin{aligned}&|\gamma_w \cap [1, i - T - 2k - 1]| - |\boldsymbol{\delta}_w \cap [1, i - T - 2k - 1]| + x'\\ =&|\gamma_{w'} \cap [1, i - T - 2k - 1]| - |\boldsymbol{\delta}_{w'} \cap [1, i - T - 2k - 1]|\end{aligned} \tag{8.15}$$

for some $x' \neq x$. If $x' > x$, then we have that

$$\begin{aligned}&c_{[i-T-k+x'-x,i-k]}\\ \overset{(a)}{=}&\boldsymbol{E}_{w,[i-T-k+x'-x+|\gamma_w\cap[1,i-T-2k-1]|-|\boldsymbol{\delta}_w\cap[1,i-T-2k-1]|,i-k+|\gamma_w\cap[1,i-T-2k-1]|-|\boldsymbol{\delta}_w\cap[1,i-T-2k-1]|]}\\ \overset{(b)}{=}&\boldsymbol{E}_{w',[i-T-k+x'+|\gamma_w\cap[1,i-T-2k-1]|-|\boldsymbol{\delta}_w\cap[1,i-T-2k-1]|,i-k+|\gamma_w\cap[1,i-T-2k-1]|-|\boldsymbol{\delta}_w\cap[1,i-T-2k-1]|+x]}\\ \overset{(c)}{=}&\boldsymbol{E}_{w',[i-T-k+|\gamma_{w'}\cap[1,i-T-2k-1]|-|\boldsymbol{\delta}_{w'}\cap[1,i-T-2k-1]|,i-k+|\gamma_{w'}\cap[1,i-T-2k-1]|-|\boldsymbol{\delta}_{w'}\cap[1,i-T-2k-1]|+x-x']}\\ \overset{(d)}{=}&c_{[i-T-k,i-k+x-x']},\end{aligned}$$

where (a) and (d) follows from (8.12) and the fact that $|\gamma_w|+|\delta_w| \leq k$ for $w \in [1, d]$, (b) follows from (8.13), and (c) follows from (8.15).

If $x' < x$, we have that

$$\begin{aligned}
&c_{[i-T-k,i-k-x+x']} \\
\overset{(a)}{=}&\boldsymbol{E}_{w,[i-T-k+|\gamma_w\cap[1,i-T-2k-1]|-|\boldsymbol{\delta}_w\cap[1,i-T-2k-1]|,i-k-x+x'+|\gamma_w\cap[1,i-T-2k-1]|-|\boldsymbol{\delta}_w\cap[1,i-T-2k-1]|]} \\
=&\boldsymbol{E}_{w',[i-T-k+|\gamma_w\cap[1,i-T-2k-1]|-|\boldsymbol{\delta}_w\cap[1,i-T-2k-1]|+x,i-k+x'+|\gamma_w\cap[1,i-T-2k-1]|-|\boldsymbol{\delta}_w\cap[1,i-T-2k-1]|]} \\
=&\boldsymbol{E}_{w',[i-T-k+|\gamma_{w'}\cap[1,i-T-2k-1]|-|\boldsymbol{\delta}_{w'}\cap[1,i-T-2k-1]|+x-x',i-k+|\gamma_{w'}\cap[1,i-T-2k-1]|-|\boldsymbol{\delta}_{w'}\cap[1,i-T-2k-1]|]} \\
\overset{(b)}{=}&c_{[i-T-k+x-x',i-k]}.
\end{aligned}$$

In both cases, we have that $L(\mathbf{c}, |x - x'|) \geq T + 1$, contradicting to the fact that $L(\mathbf{c}, \leq k) \leq T$. Hence, $x' = x$ and the proof is done. $\square$

Correcting $k < d$ Deletions and Insertions

The cases when $k < d$ are addressed in the following lemma, which proves the first part of Theorem 8.1.2, where $k < d$.

Lemma 8.6.2. *Let $\boldsymbol{E} \in \mathcal{E}_k(\mathbf{c})$ be a read matrix for some sequence $\mathbf{c}$ satisfying $L(\mathbf{c}, \leq k) \leq T$. Let the distance t satisfy $t > (\frac{k^2}{4} + 3k + 2)(T + 3k + 1) + T + 5k + 1$. If there is an interval $[b_1, b_2]$, an interval $[p_1, p_2] \subseteq [b_1, b_2]$, and an edit isolated interval $\mathcal{I}$ satisfying $\boldsymbol{E}_{[1,d],[p_1,p_2]} \in \mathcal{E}_{k'}(\mathbf{c}_{\mathcal{I}})$ for some $k' \leq d - 1$, and $\boldsymbol{E}_{w,j} = \boldsymbol{E}_{w',j}$ for any $w, w' \in [1, d]$ and $j \in ([b_1, p_1 - 1] \cup [p_2 + 1, b_2])$, then we can obtain a read matrix $\boldsymbol{E}'$ such that $\boldsymbol{E}'_{[1,d],[p_1,p_2]} \in \mathcal{E}_0(\mathbf{c}_{\mathcal{I}})$, and $\boldsymbol{E}_{w,j} = \boldsymbol{E}_{w',j}$ for any $w, w' \in [1, d]$ and $j \in ([b_1, p_1 - 1] \cup [p_2 + 1, b_2])$.*

Remark 8.6.1. *Lemma 8.6.2 is a generalization of and an improvement over Lemma 8.2.6 when $k \leq d$ and k is sufficiently large.*

Proof. Let i^* be the minimum index such that $i^* > p_1$ and there exist different $w, w' \in [1, d]$ satisfying $\boldsymbol{E}_{w,i^*} \neq \boldsymbol{E}_{w',i^*}$. Let $\boldsymbol{E}_{w_1,i^*}$ be the minority bit among $\{\boldsymbol{E}_{w,i^*}\}_{w=1}^{d}$, i.e., there are at most $\lfloor \frac{d}{2} \rfloor$ bits among $\{\boldsymbol{E}_{w,i^*}\}_{w=1}^{d}$ being equal to $\boldsymbol{E}_{w,i^*} \neq \boldsymbol{E}_{w',i^*}$. We will first show that there are edit errors occur near index i^* in the w_1-th head, unless when the numbers of 1-bits and 0-bits among $\{\boldsymbol{E}_{w,i^*}\}_{w=1}^{d}$ are equal, edit errors occur near index i^* in the first head. To this end, we begin with the following proposition.

Proposition 8.6.2. *Let $\boldsymbol{E} \in \mathcal{E}_k(\mathbf{c})$ be a read matrix for some sequence $\mathbf{c}$ satisfying $L(\mathbf{c}, \leq k) \leq T$. Let $i^* > 0$ be an integer such that $\boldsymbol{E}_{w,[i^*-T-2k-1,i^*-1]} = \boldsymbol{E}_{w',[i^*-T-2k-1,i^*-1]}$ for any $w', w \in [1, d]$. For any $w, w' \in [1, d]$ such that no error* *occurs in interval* $[i^* - T - 2k, i^* + k - 1]$ *in the* w*-th and* w'*-th head, i.e.,*

$$\begin{aligned}&(\boldsymbol{\delta}_w \cup \boldsymbol{\gamma}_w) \cap [i^* - T - 2k - 1, i^* + k - 1] = \emptyset, \ \textit{and}\\ &(\boldsymbol{\delta}_{w'} \cup \boldsymbol{\gamma}_{w'}) \cap [i^* - T - 2k - 1, i^* + k - 1] = \emptyset,\end{aligned} \tag{8.16}$$

the bits $\boldsymbol{E}_{w,i^*}$ *and* $\boldsymbol{E}_{w',i^*}$ *are equal.*

Proof. According to Proposition 8.6.1, we have that

$$\begin{aligned}&|\boldsymbol{\gamma}_w \cap [1, i^* - T - 2k - 2]| - |\boldsymbol{\delta}_w \cap [1, i^* - T - 2k - 2]|\\ =&|\boldsymbol{\gamma}_{w'} \cap [1, i^* - T - 2k - 2]| - |\boldsymbol{\delta}_{w'} \cap [1, i^* - T - 2k - 2]|\end{aligned} \tag{8.17}$$

for any $w, w' \in [1, d]$. Then,

$$\begin{aligned}\boldsymbol{E}_{w,i^*} &\overset{(a)}{=} \mathbf{c}_{i^* - |\boldsymbol{\gamma}_w \cap [1, i^* - T - 2k - 2]| + |\boldsymbol{\delta}_w \cap [1, i^* - T - 2k - 2]|}\\ &\overset{(b)}{=} \mathbf{c}_{i^* - |\boldsymbol{\gamma}_{w'} \cap [1, i^* - T - 2k - 2]| + |\boldsymbol{\delta}_{w'} \cap [1, i^* - T - 2k - 2]|}\\ &\overset{(c)}{=} \boldsymbol{E}_{w,i^*},\end{aligned}$$

where (a) and (c) follow from (8.16) and the fact that $|\boldsymbol{\gamma}_w \cap [1, i^* - T - 2k - 2]| - |\boldsymbol{\delta}_w \cap [1, i^* - T - 2k - 2]| \leq k$. Equality (b) follows from (8.17). □

From Proposition 8.6.2, we can easily conclude that there is at least one error in interval $[i^* - T - 2k - 1, i^* + k - 1]$ in one of the heads, i.e., $(\boldsymbol{\delta}_w \cup \boldsymbol{\gamma}_w) \cap [i^* - T - 2k - 1, i^* + k - 1] \neq \emptyset$ for some $w \in [1, d]$. Otherwise the bits $\boldsymbol{E}_{w,i^*}$ are equal for all $w \in [1, d]$, contradicting to the definition of i^*.

Next, we need the following proposition.

Proposition 8.6.3. *Let* $\boldsymbol{E} \in \mathcal{E}_k(\mathbf{c})$ *be a read matrix for some sequence* $\mathbf{c}$ *satisfying* $L(\mathbf{c}, \leq k) \leq T$. *Let* $i^* > 0$ *be an integer such that* $\boldsymbol{E}_{w,[1,i^*-1]} = \boldsymbol{E}_{w',[1,i^*-1]}$ *for any* $w', w \in [1, d]$.*If* $T^* \geq T + 2k + 1$ *and* $t > (k+1)T^*$*, then the number of heads where at least one error occurs in interval* $[i^* - T^*, i^* + k - 1]$ *is at most* $\lfloor \frac{k+1}{2} \rfloor$*, i.e.,*

$$|\{w : (\boldsymbol{\delta}_w \cup \boldsymbol{\gamma}_w) \cap [i^* - T^*, i^* + k - 1] \neq \emptyset\}| \leq \lfloor \frac{k+1}{2} \rfloor.$$

Moreover, when $|\{w : (\boldsymbol{\delta}_w \cup \boldsymbol{\gamma}_w) \cap [i^* - T^*, i^* + k - 1] \neq \emptyset\}| = \frac{k+1}{2}$*, at least one error occurs in* $[i^* - T^*, i^*]$ *in the first head, i.e.,* $(\boldsymbol{\delta}_1 \cup \boldsymbol{\gamma}_1) \cap [i^* - T^*, i^*] \neq \emptyset$.

Proof. Let $\{w : w \in [2, d], (\boldsymbol{\delta}_w \cup \boldsymbol{\gamma}_w) \cap [i^* - T^*, i^* + k - 1] \neq \emptyset\} = \{w_1, w_2, \ldots, w_M\}$ be the set of heads (not including the first head) that contains at least one error in interval $[i^* - T^*, i^* + k - 1]$. Let $w_1 > w_2 > \ldots > w_M$. We will show that there exist a set of integers $i_1, i_2, \ldots, i_M \in [0, k]$ such that $i_1 \geq i_2 \geq \ldots \geq i_M$ and

$$\begin{aligned}&|(\boldsymbol{\delta}_1 \cap [i^* - T^* - (w_\ell - 1)t - (T^* + k)i_\ell, i^* - T^* - (w_\ell - 2)t - (T^* + k)i_\ell - 1]| \\ &\quad + |\gamma_1 \cap [i^* - T^* - (w_\ell - 1)t - (T^* + k)i_\ell, i^* - T^* - (w_\ell - 2)t - (T^* + k)i_\ell - 1]| \\ \geq& 2 \end{aligned} \tag{8.18}$$

for $\ell \in [1, M]$. Note that the intervals $[i^* - T^* - (w_\ell - 1)t - (T^* + k)i_\ell, i^* - T^* - (w_\ell - 2)t - (T^* + k)i_\ell - 1]$ are disjoint for different $\ell \in [1, M]$ and are within the interval $[-T^* - (T^* + k)(k + 1), i^* - T^* - 1]$, since $t_j > (k + 1)(T^* + 1)$ for $j \in [1, d]$ and $i_\ell \leq k$ for $\ell \in [1, M]$. Then, the number of errors in the first head is at least $2|\{w : (\boldsymbol{\delta}_w \cup \gamma_w) \cap [i^* - T^*, i^* + k - 1] \neq \emptyset, w \in [2, d]\}| + \mathbb{1}((\boldsymbol{\delta}_1 \cup \gamma_1) \cap [i^* - T^*, i^* + k - 1] \neq \emptyset)$, where $\mathbb{1}(A)$ is the indicator that equals 1 when A is true and equals 0 otherwise. Hence, we have that

$$\begin{aligned}&2|\{w : (\boldsymbol{\delta}_w \cup \gamma_w) \cap [i^* - T^*, i^* + k - 1] \neq \emptyset, w \in [2, d]\}| \\ &+ \mathbb{1}((\boldsymbol{\delta}_1 \cup \gamma_1) \cap [i^* - T^*, i^* + k - 1] \neq \emptyset) \leq k.\end{aligned}$$

Then, it can be easily verified that the proposition follows.

Now we find the set of integers $i_1 \geq i_2 \geq \ldots \geq i_M$ satisfying (8.18). Let $i_0 = k$. Starting from $\ell = 1$ to $\ell = M$, find the largest integer i_ℓ such that $i_\ell \leq i_{\ell-1}$ and no errors occur in interval $[i^* - T^* - (T^* + k)(i_\ell + 1), i^* - T^* - (T^* + k)i_\ell - 1]$ in the w_ℓ-th or the $(w_\ell - 1)$-th heads, i.e.,

$$\begin{aligned}&(\gamma_{w_\ell} \cup \boldsymbol{\delta}_{w_\ell} \cup \gamma_{w_\ell - 1} \cup \boldsymbol{\delta}_{w_\ell - 1}) \cap [i^* - T^* - (T^* + k)(i_\ell + 1), \\ &i^* - T^* - (T^* + k)i_\ell - 1] = \emptyset.\end{aligned} \tag{8.19}$$

We show that such an $\ell \in [1, M]$ can be found as long as $t > (T^* + k)(k + 2)$. Note that in the above procedure, for each integer $i \in [i_\ell + 1, k]$, there is at least an edit error occurring in interval $[i^* - T^* - (T^* + k)(i + 1), i^* - T^* - (T^* + k)i - 1]$ in one of the heads w, which corresponds to an error that occurs in interval $[i^* - T^* - (T^* + k)(i + 1) - (w - 1)t, i^* - T^* - (T^* + k)i - 1 - (w - 1)t]$ in the first head. In addition, the intervals $[i^* - T^* - (T^* + k)(i + 1) - (w - 1)t, i^* - T^* - (T^* + k)i - 1 - (w - 1)t]$ are disjoint for different pairs (i, w), as long as $t \geq (T^* + k)(k + 2)$. Since there are at most k errors in the first head and there are $k + 1$ choices of i_ℓ, such an i_ℓ satisfying (8.19) can be found.

Since $\boldsymbol{E}_{w_\ell, i} = \boldsymbol{E}_{w_\ell - 1, i}$ for $i \in [i^* - T^* - (T^* + k)(i_\ell + 1), i^* - T^* - (T^* + k)i_\ell - 1]$, by Proposition 8.6.1 we have that

$$|\gamma_{w_\ell - 1} \cap [1, i^* - T^* - (T^* + k)(i_\ell + 1) - 1]|$$

$$\begin{aligned}&-|\boldsymbol{\delta}_{w_\ell-1}\cap[1,i^*-T^*-(T^*+k)(i_\ell+1)-1]|\\=&|\gamma_{w_\ell}\cap[1,i^*-T^*-(T^*+k)(i_\ell+1)-1]|\\&-|\boldsymbol{\delta}_{w_\ell}\cap[1,i^*-T^*-(T^*+k)(i_\ell+1)-1]|.\end{aligned}\tag{8.20}$$

On the other hand, we have that

$$\begin{aligned}&|\gamma_{w_\ell-1}\cap[1,i^*-T^*-(T^*+k)(i_\ell+1)-1-t]|\\&-|\boldsymbol{\delta}_{w_\ell-1}\cap[1,i^*-T^*-(T^*+k)(i_\ell+1)-1-t]|\\=&|\gamma_{w_\ell}\cap[1,i^*-T^*-(T^*+k)(i_\ell+1)-1]|\\&-|\boldsymbol{\delta}_{w_\ell}\cap[1,i^*-T^*-(T^*+k)(i_\ell+1)-1]|.\end{aligned}\tag{8.21}$$

Eq. (8.20) and Eq. (8.21) imply that

$$\begin{aligned}&|\gamma_{w_\ell-1}\cap[i^*-T^*-(T^*+k)(i_\ell+1)-t,i^*-T^*-(T^*+k)(i_\ell+1)-1]|\\=&|\boldsymbol{\delta}_{w_\ell-1}\cap[i^*-T^*-(T^*+k)(i_\ell+1)-t,i^*-T^*-(T^*+k)(i_\ell+1)-1]|.\end{aligned}\tag{8.22}$$

Since $(\gamma_{w_\ell}\cup\boldsymbol{\delta}_{w_\ell})\cap[i^*-T^*,i^*+k-1]\neq\emptyset$ by definition of w_ℓ, we have that

$$\begin{aligned}&(\gamma_{w_\ell-1}\cup\boldsymbol{\delta}_{w_\ell-1})\cap[i^*-T^*-t,i^*+k-1-t]\\\subseteq&(\gamma_{w_\ell-1}\cup\boldsymbol{\delta}_{w_\ell-1})\cap[i^*-T^*-(T^*+k)(i_\ell+1)-t,i^*-T^*-(T^*+k)(i_\ell+1)-1]\\\neq&\emptyset.\end{aligned}$$

Together with (8.22), we have that

$$\begin{aligned}&|\gamma_{w_\ell-1}\cap[i^*-T^*-(T^*+k)(i_\ell+1)-t,i^*-T^*-(T^*+k)(i_\ell+1)-1]|\\+&|\boldsymbol{\delta}_{w_\ell-1}\cap[i^*-T^*-(T^*+k)(i_\ell+1)-t,i^*-T^*-(T^*+k)(i_\ell+1)-1]|\\\geq&2,\end{aligned}$$

which implies (8.18) because $\gamma_{w_\ell-1}=\gamma_1+(w_\ell-2)t$ and $\boldsymbol{\delta}_{w_\ell-1}=\boldsymbol{\delta}_1+(w_\ell-2)t$. Hence, the proof is done. □

Let $w^*\in[1,d]$ be an index of the head such that $\boldsymbol{E}_{w^*,i^*}$ is a minority bit among $\{\boldsymbol{E}_{w,i^*}\}_{w=1}^d$, i.e., there are at most $\frac{d}{2}$ bits among $\{\boldsymbol{E}_{w,i^*}\}_{w=1}^d$ that is equal to $\boldsymbol{E}_{w^*,i^*}$. By Proposition 8.6.2 and Proposition 8.6.3, we conclude that when $k<d$, we have that $(\boldsymbol{\delta}_{w^*}\cup\gamma_{w^*})\cap[i^*-T-2k-1,i^*+k-1]\neq\emptyset$, if the number of bits among $\{\boldsymbol{E}_{w,i^*}\}_{w=1}^d$ being equal to $\boldsymbol{E}_{w^*,i^*}$ is is less than $d/2$. If $k<d$ and the number of bits among $\{\boldsymbol{E}_{w,i^*}\}_{w=1}^d$ being equal to $\boldsymbol{E}_{w^*,i^*}$ is is exactly $d/2$, we have that $(\boldsymbol{\delta}_1\cup\gamma_1)\cap[i^*-T-2k-1,i^*+k-1]\neq\emptyset$.

Now we find a w^* with $(\boldsymbol{\delta}_{w^*} \cup \gamma_{w^*}) \cap [i^* - T - 2k - 1, i^* + k - 1] \neq \emptyset$. In the remaining part of the proof, we show how to use knowledge of w^* to correct at least one error for each head, and reduce the d-head case to a $(d-1)$-head case. Then, the lemma follows by induction. Assume that $w^* \leq d - 1$. The procedure when $w^* = d$ will be similar.

Note that $(\boldsymbol{\delta}_w \cup \gamma_w) \cap [i^* - T - 2k - 1 + (w - w^*)t, i^* + k - 1 + (w - w^*)t] \neq \emptyset$. Consider the set of intervals

$$[i^* + 2k + (\ell - 1)(T + 3k + 1) + (w - w^*)t,$$
$$i^* + 2k - 1 + \ell(T + 3k + 1) + (w - w^*)t]$$

for $\ell \in [1, \frac{k^2}{4} + 3k + 2]$ and $w \in [1, d]$. For notation convenience, denote

$$v_{w,\ell} \triangleq i^* + 2k + (\ell - 1)(T + 3k + 1) + (w - w^*)t \tag{8.23}$$

for $\ell \in [1, \frac{k^2}{4} + 3k + 2]$ and $w \in [1, d]$. For each pair $\ell \in [1, \frac{k^2}{4} + 3k + 2]$ and $w \in [1, d - 1]$, find a unique index $x_{w,\ell} \in [0, k]$, such that

$$\boldsymbol{E}_{w,[v_{w,\ell}, v_{w,\ell+1} - 1 - x_{w,\ell}]} = \boldsymbol{E}_{w+1,[v_{w,\ell} + x_{w,\ell}, v_{w,\ell+1} - 1]} \tag{8.24}$$

or $x_{w,\ell} \in [-k, -1]$ such that

$$\boldsymbol{E}_{w,[v_{w,\ell} - x_{w,\ell}, v_{w,\ell} + T + 3k]} = \boldsymbol{E}_{w+1,[v_{w,\ell}, v_{w,\ell} + x_{w,\ell} + + T + 3k]}. \tag{8.25}$$

If no such index or more than one exist, let $x_{w,\ell} = k + 1$.

Given $x_{w,\ell}$, $\ell \in [1, \frac{k^2}{4} + 3k + 2]$ and $w \in [1, d]$, define a binary vector $\mathbf{z} \in \{0, 1\}^{\frac{k^2}{4} + 3k + 2}$ as follows:

$$z_\ell = \begin{cases} 1, \text{if there exists a } w \in [1, d-1] \text{ such that } x_{w,\ell} = k + 1 \\ 1, \text{if there exists a } w \in [1, d-1] \text{ such that } x_{w,\ell} \neq x_{w,\ell-1} \\ \quad \text{and } x_{w,\ell}, x_{w,\ell-1} \in [-k, k] \\ 0, \text{else} \end{cases} \tag{8.26}$$

for $\ell \in \frac{k^2}{4} + 3k + 2$. In (8.26), it is assumed that $x_{w,0} = x_{w,1}$ for $w \in [1, d - 1]$.

Let y be the number of 1 runs in $\mathbf{z}$. Let $y^* = |(\gamma_{w^*} \cup \boldsymbol{\delta_{w^*}} \cup \gamma_{w^*+1} \cup \boldsymbol{\delta_{w^*+1}}) \cap [i^* + k, i^* + 3k - 1 + (\frac{k^2}{4} + 3k + 2)(T + 3k + 1)]|$ be the number of errors that occur in interval $[i^* + k, i^* + k - 1 + (\frac{k^2}{4} + 3k + 2)(T + 3k + 1)]$ in the w^*-th or $(w^* + 1)$-th head. Note that $y^* = |(\gamma_w \cup \boldsymbol{\delta_w} \cup \gamma_{w+1} \cup \boldsymbol{\delta_{w+1}}) \cap [i^* + k + (w -$

$w^*)t, i^*+3k-1+(\frac{k^2}{4}+3k+2)(T+3k+1)+(w-w^*)t]|$ for $w \in [1,d]$. Moreover, $\boldsymbol{E}_{w,[i^*+2k+(w-w^*)t, i^*+2k-1+(\frac{k^2}{4}+3k+2)(T+3k+1)+(w-w^*)t]}$ can be obtained by a subsequence of $\mathbf{c}_{[i^*+k, i^*+3k-1+(\frac{k^2}{4}+3k+2)(T+3k+1)]}$ after at most y^* deletions and insertions in interval $[i^*+k, i^*+3k-1+(\frac{k^2}{4}+3k+2)(T+3k+1)]$ in the w-th head, $w \in [1,d]$.

We first show that $y^* \leq k-1$. Note that the $|(\gamma_w \cup \boldsymbol{\delta_w}) \cap [i^*+k, n+k]|$ errors that occur after index i^*+k in the w^*-th head, occur after index $i^*+k+t > i^*+3k+(\frac{k^2}{4}+3k+2)(T+3k+1)$ in the (w^*+1)-th head. Moreover, the errors that occur in interval $[i^*-T-2k-1, i^*+k-1]$ in the w^*-th head occur after $i^*-T-2k-1+t > i^*+3k+(\frac{k^2}{4}+3k+2)(T+3k+1)$ in the (w^*+1)-th head, since $t > (\frac{k^2}{4}+3k+2)(T+3k+1)+T+5k+1$. Hence there are at most $k-|(\gamma_{w^*} \cup \boldsymbol{\delta_{w^*}}) \cap [i^*+k, n+k]|-1+|(\gamma_{w^*} \cup \boldsymbol{\delta_{w^*}}) \cap [i^*+k, n+k]| = k-1$ errors that occur in interval $[i^*+k, i^*+3k-1+(\frac{k^2}{4}+3k+2)(T+3k+1)]$ in the w^*-th or (w^*+1)-th head.

Next, we show that there are at most $(2k-2)$ 1 entries in $\mathbf{z}$. Note that a single error in interval $[i^*+k, i^*+3k-1+(\frac{k^2}{4}+3k+2)(T+3k+1)]$ in the w-th or $(w+1)$-th head affects the value of at most a single entry $x_{w,\ell}$ and the entries $(x_{w,\ell+1}, \ldots, x_{w,\frac{k^2}{4}+3k+2}$ increase or decrease by 1. This generates at most two 1 entries in $\mathbf{z}$. Hence there are at most $2y^* \leq 2k-2$ 1 entries in $\mathbf{z}$.

Then, we show that there exists a 0-run $(z_{i+1}, \ldots, z_{i+k-y+2})$ of length $k-y+2$, for some $i \in [0, \frac{k^2}{4}+2k+y]$, which indicates that

$$\boldsymbol{E}_{w,[v_{w,i+1}, v_{w,i+k-y+3}-x_{w,i+1}-1]} = \boldsymbol{E}_{w+1,[v_{w,i+1}+x_{w,i+1}, v_{w,i+k-y+3}-1]}, \tag{8.27}$$

if $x_{w,i+1} \in [0,k]$ or

$$\boldsymbol{E}_{w,[v_{w,i+1}-x_{w,i+1}, v_{w,i+k-y+3}-1]} = \boldsymbol{E}_{w+1,[v_{w,i+1}, v_{w,i+k-y+3}+x_{w,i+1}-1]}, \tag{8.28}$$

if $x_{w,i+1} \in [-k,-1]$, for every $w \in [1, d-1]$.

Suppose on the contrary, each 0 run has length no more than $k-y+1$. Note that there are at most $y+1$ 0 runs with y 1 runs. Therefore, the length of $\mathbf{z}$ is upper bounded by

$$\begin{aligned} \frac{k^2}{4}+3k+2 \leq& (y+1)(k-y+1)+2k \\ =& -y^2+ky+3k+1 \\ \leq& \frac{k^2}{4}+3k+1 \end{aligned}$$

a contradiction.

We have proved the existence of a 0 run $(z_i+1,\dots,z_{i+k-y+2})$, which implies (8.27) and (8.28). We now show that there are at most $k-y+1$ errors occur in interval $[v_{w,i+1}, v_{w,i+k-y+3}-1]$ in the w and/or $(w+1)$-th head, for $w\in[1,d-1]$. As mentioned above, a single error in interval $[i^*+k, i^*+3k-1+(\frac{k^2}{4}+3k+2)(T+3k+1)]$ in the w-th or $(w+1)$-th head affects the value of at most a single entry $x_{w,\ell}$ and the entries $(x_{w,\ell+1},\dots,x_{w,\frac{k^2}{4}+3k+2})$ increase or decrease by 1. This generates at most a single 1 run in $\mathbf{z}$. In addition, errors in interval $[v_{w,i+1}, v_{w,i+k-y+3}-1]$ in the w and/or $(w+1)$-th head generate at most two 1 runs that include z_i and $z_{i+k-y+2}$. Therefore, there are at least $y-2$ 1 runs in $\mathbf{z}$ that are generated by at least $y-2$ errors in $[i^*+k, i^*+3k-1+(\frac{k^2}{4}+3k+2)(T+3k+1)]\backslash[v_{w,i+1}, v_{w,i+k-y+3}-1]$. Hence, the number of errors in interval $[v_{w,i+1}, v_{w,i+k-y+3}-1]$ in the w and/or $(w+1)$-th head is at most $y^*-y+2\le k-y+1$.

Therefore, there exists an integer $\ell\in[i+1,i+k-y+2]$ such that no errors occur in interval $[v_{w,\ell}, v_{w,\ell+1}-1]$ in the w and/or $(w+1)$-th head, which implies that $\boldsymbol{E}_{w+1,[p_1,v_{w,\ell}-|\boldsymbol{\delta}_{w+1}\cap\mathcal{I}\cap[1,v_{w,\ell}-1]|+|\boldsymbol{\gamma}_{w+1}\cap\mathcal{I}\cap[1,v_{w,\ell}-1]|+k]}$ is obtained from $\mathbf{c}_{\mathcal{I}\cap[1,v_{w,\ell}+k]}$, after deletion errors at locations $\boldsymbol{\delta}_{w+1}\cap\mathcal{I}\cap[1,v_{w,\ell}-1]$ and insertion errors at locations $\boldsymbol{\gamma}_{w+1}\cap\mathcal{I}\cap[1,v_{w,\ell}-1]$. Moreover,

$$\begin{aligned}&\boldsymbol{E}_{w,[v_{w,\ell}+k+1-|\boldsymbol{\delta}_w\cap\mathcal{I}\cap[1,v_{w,\ell}-1]|+|\boldsymbol{\gamma}_w\cap\mathcal{I}\cap[1,v_{w,\ell}-1]|,p_2]}\\ =&\boldsymbol{E}_{w,[v_{w,\ell}+k+1-|\boldsymbol{\delta}_{w+1}\cap\mathcal{I}\cap[1,v_{w,\ell}-1]|+|\boldsymbol{\gamma}_{w+1}\cap\mathcal{I}\cap[1,v_{w,\ell}-1]|+k-x_{w,\ell},p_2]}\\ =&\boldsymbol{E}_{w,[v_{w,\ell}+k+1-|\boldsymbol{\delta}_{w+1}\cap\mathcal{I}\cap[1,v_{w,\ell}-1]|+|\boldsymbol{\gamma}_{w+1}\cap\mathcal{I}\cap[1,v_{w,\ell}-1]|+k-x_{w,i+1},p_2]}\end{aligned}$$

can be obtained from $\mathbf{c}_{\mathcal{I}\cap[v_{w,\ell}+k+1,n+k]}$, after deletion errors at locations $\boldsymbol{\delta}_w\cap\mathcal{I}\cap[v_{w,\ell},n+k]$ and insertion errors at locations $\boldsymbol{\gamma}_w\cap\mathcal{I}\cap[v_{w,\ell},n+k]$. Therefore, concatenating $\boldsymbol{E}_{w+1,[p_1,v_{w,\ell}-|\boldsymbol{\delta}_{w+1}\cap\mathcal{I}\cap[1,v_{w,\ell}-1]|+|\boldsymbol{\gamma}_{w+1}\cap\mathcal{I}\cap[1,v_{w,\ell}-1]|+k]}$ and $\boldsymbol{E}_{w,[v_{w,\ell}+k+1-|\boldsymbol{\delta}_{w+1}\cap\mathcal{I}\cap[1,v_{w,\ell}-1]|+|\boldsymbol{\gamma}_{w+1}\cap\mathcal{I}\cap[1,v_{w,\ell}-1]|+k-x_{w,i+1},p_2]}$, we have a sequence, obtained from $\mathbf{c}_{\mathcal{I}}$ by deletion errors with locations $(\boldsymbol{\delta}_{w+1}\cap[1,v_{w,\ell}-1])\cup(\boldsymbol{\delta}_w\cap[v_{w,\ell},n+k])$ and insertion errors at locations $(\boldsymbol{\gamma}_{w+1}\cap[1,v_{w,\ell}-1])\cup(\boldsymbol{\gamma}_w\cap[v_{w,\ell},n+k])$, $w\in[1,d-1]$ in $\mathbf{c}_{\mathcal{I}}$. Note that there are at most $|\boldsymbol{\delta}_w\cap\mathcal{I}|+|\boldsymbol{\gamma}_w\cap\mathcal{I}|-1$ errors in total in the concatenation, since the errors occur in $[(\boldsymbol{\delta}_w\cup|\boldsymbol{\gamma}_w)\cap[i^*-T-2k-1+(w-w^*)t, i^*+k-1+(w-w^*)t]|]$ are not included in the concatenation. Finally, since $(z_{i+1},\dots,z_{i+k-y+2})$ is a 0 run, we have that $x_{w,i+1}=x_{w,i+2}=\dots=x_{w,i+k-y+2}$ for $w\in[1,d-1]$. Hence, concatenating

$\boldsymbol{E}_{w+1,[b_1,v_{w,i+1}+k]}$ and $\boldsymbol{E}_{w,[v_{w,i+1}+k+1-x_{w,i+1},b_2]}$ results in the same sequence as concatenating $\boldsymbol{E}_{w+1,[b_1,v_{w,\ell}-|\boldsymbol{\delta}_{w+1}\cap\mathcal{I}\cap[1,v_{w,\ell}-1]|+|\boldsymbol{\gamma}_{w+1}\cap\mathcal{I}\cap[1,v_{w,\ell}-1]|+k]}$ and

$$\boldsymbol{E}_{w,[v_{w,\ell}+k+1-|\boldsymbol{\delta}_{w+1}\cap\mathcal{I}\cap[1,v_{w,\ell}-1]|+|\boldsymbol{\gamma}_{w+1}\cap\mathcal{I}\cap[1,v_{w,\ell}-1]|+k-x_{w,i+1},b_2]}, w \in [1, d-1].$$

Let the $d-1$ concatenated sequences be represented by a read matrix $\boldsymbol{E}'$. Then there exists some interval $[p_1', p_2']$ such that $\boldsymbol{E}'_{[1,d-1],[p_1',p_2']} \in \mathcal{E}_{k'-1}(\mathbf{c}_{\mathcal{I}})$. In addition, we have that $\boldsymbol{E}'_{w,j} = \boldsymbol{E}'_{w',j}$ for any $w, w' \in [1, d-1]$ and $j \in ([1, p_1'-1] \cup [p_2'+1, b'])$, where b' is the number of columns in $\boldsymbol{E}'$. Then, we obtain a read matrix with $d-1$ head and less errors. The lemma follows by induction.

For cases when $w^* = d$, the proof is similar, where instead of looking at intervals $[i^*+2k+(\ell-1)(T+3k+1)+(w-w^*)t, i^*+2k-1+\ell(T+3k+1)+(w-w^*)t]$ for $\ell \in [1, \frac{k^2}{4}+3k+2]$ and $w \in [1, d]$, we look at intervals $[i^*-T-3k-\ell(T+3k+1)+(w-w^*)t, i^*-T-3k-1-(\ell-1)(T+3k+1)+(w-w^*)t]$ for $\ell \in [1, \frac{k^2}{4}+3k+2]$ and $w \in [1, d]$.

□

Determine Bits Outside Edit Isolated Intervals

Lemma 8.6.3. *For any edit isolated interval $\mathcal{I}$ and an interval $[p_1, p_2]$ such that $\boldsymbol{E}_{[1,d],[p_1,p_2]} \in \mathcal{E}_{k'}(\mathbf{c}_{\mathcal{I}})$ for some sequence $\mathbf{c}$ satisfying $L(\mathbf{c}, \leq k) \leq T$, if $\boldsymbol{E}_{w,j} = \boldsymbol{E}_{w',j}$ for any $w, w' \in [1, d]$ and $j \in ([p_1, p_2])$, then $|\boldsymbol{\delta}_w \cap \mathcal{I}| + |\boldsymbol{\gamma}_w \cap \mathcal{I}| \geq 2d$.*

Proof. Let $\mathcal{I} = [q_1, q_2]$. Let i^* be the minimum index such that $i^* \geq q_1$ and $\boldsymbol{E}_{w,i^*} \neq \mathbf{c}_{q_1+i^*-p_1}$, i.e., $\boldsymbol{E}_{w,[p_1,i^*-1]} = \mathbf{c}_{[q_1,q_1+i^*-p_1-1]}$ and $\boldsymbol{E}_{w,i^*} \neq \mathbf{c}_{q_1+i^*-p_1}$ for $w \in [1, d]$. We show that $(\boldsymbol{\delta}_w \cup \boldsymbol{\gamma}_w) \cap [i^*-T-2k-1, i^*+k-1] \neq \emptyset$ for $w \in [1, d]$. Otherwise, there exists a $w^* \in [1, d]$, such that $(\boldsymbol{\delta}_{w^*} \cup \boldsymbol{\gamma}_{w^*}) \cap [i^*-T-2k-1, i^*+k-1] = \emptyset$. Assume now that there is a virtual read $\boldsymbol{E}_{w',[1,r]}$, where r is the length of each read in the first d heads. Assume that the distance between the d-th head and the w'-th head is far enough so that the first error occurs after index $q_1+i^*-p_1$ in the w'-th head. Therefore, $\boldsymbol{E}_{w',[p_1,i^*-1]} = \mathbf{c}_{[q_1,q_1+i^*-p_1-1]} = \boldsymbol{E}_{w,[p_1,i^*-1]}$ for $w \in [1, d]$ and $\boldsymbol{E}_{w',i^*} = \mathbf{c}_{q_1+i^*-p_1}$. By Proposition 8.6.2, we have that $\boldsymbol{E}_{w^*,i^*} = \mathbf{c}_{q_1+i^*-p_1}$, contradicting to the definition of i^*. Hence, we have that $(\boldsymbol{\delta}_w \cup \boldsymbol{\gamma}_w) \cap [i^*-T-2k-1, i^*+k-1] \neq \emptyset$ for $w \in [1, d]$.

According to Proposition 8.6.3, we have that $k \geq 2d-1$. Furthermore, by Lemma 8.6.4, we have that k is an odd number and thus $k \geq 2d$. □

Lemma 8.6.4. *Let $\boldsymbol{E} \in \mathcal{E}_k(\mathbf{c})$ be a read matrix for some sequence $\mathbf{c}$ satisfying $L(\mathbf{c}, \leq k) \leq T$. Let the head distance t satisfy $t > (4k+1)(T+4k+1)$. If there is an interval $[b_1, b_2]$, an interval $[p_1, p_2] \subseteq [b_1, b_2]$, and an edit isolated interval $\mathcal{I}$ satisfying $\boldsymbol{E}_{[1,d],[p_1,p_2]} \in \mathcal{E}_{k'}(\mathbf{c}_{\mathcal{I}})$ for some $k' \leq d-1$, and $\boldsymbol{E}_{w,j} = \boldsymbol{E}_{w',j}$ for any $w, w' \in [1,d]$ and $j \in ([b_1, p_1 - 1] \cup [p_2 + 1, b_2])$, then the the number of bit shifts caused by errors in interval $\mathcal{I}$, $|\gamma_w \cap \mathcal{I}| - |\boldsymbol{\delta}_w \cap \mathcal{I}|$ can be decided from $\boldsymbol{E}_{[1,d],[b_1,b_2]}$, for $w \in [1,d]$. Moreover, if $\boldsymbol{E}_{w,[b_1,b_2]} = \boldsymbol{E}_{w',[b_1,b_2]}$ for any $w, w' \in [1,d]$, then $|\gamma_w \cap \mathcal{I}| = |\boldsymbol{\delta}_w \cap \mathcal{I}|$ for any $w \in [1,d]$.*

Proof. Similar to what we did in Sec. 8.4. consider a set of intervals

$$\mathcal{B}_{i,m} = \begin{cases} [b_1 + (i-1)t + (m-1)(T+4k+1), b_1 + (i-1)t + m(T+4k+1) - 1], \\ \text{for } m \in [1, 4k+1] \text{ and } i \in [0, \frac{b_2 - b_1 + 1}{t} + 1] \text{ satisfying} \\ b_1 + (i-1)t + m(T+4k+1) - 1 \leq b_2. \end{cases}.$$

Note that the intervals $\mathcal{B}_{i,m}$ are disjoint when $t > (4k+1)(T+4k+1)$ For notation convenience, let

$$q_{i,m} \triangleq b_1 + (i-1)t + (m-1)(T+4k+1)$$

for $m \in [1, 4k+1]$ and $i \in [0, \frac{b_2-b_1+1}{t} + 1]$ satisfying $b_1 + (i-1)t + m(T+4k+1) - 1 \leq b_2$. Let $\mathcal{U}_m = \cup_{i: q_{i,m} - 1 \leq b_2, i \in [1, \frac{b_2 - b_1 + 1}{t} + 1]} \mathcal{B}_{i,m}$, for $m \in [1, 4k+1]$. Since there are at most $2k$ errors in the first two heads, there are at least $(2k+1)$ choices of $m \in [1, 4k+1]$, $m_1, \ldots, m_{2k+1}$, such that $\mathcal{U}_{m_\ell} \cap (\boldsymbol{\delta}_1 \cup \gamma_1 \cup \boldsymbol{\delta}_2 \cup \boldsymbol{\delta}_2) \cap \mathcal{I} = \emptyset$ for $\ell \in [1, 2k+1]$. For each $m \in [1, 4k+1]$ and integer $i \geq 1$ such that $q_{i,m} - 1 \leq b_2$, find the unique integer $x_{m,i} \in [0,k]$ such that

$$\boldsymbol{E}_{1,[q_{i,m+1}+k, q_{i,m+2}-k-1-x_{m,i}]} = \boldsymbol{E}_{2,[q_{i,m+1}+k+x_{m,i}, q_{i,m+2}-k-1]} \tag{8.29}$$

or $x_{m,i} \in [-k, -1]$ such that

$$\boldsymbol{E}_{1,[q_{i,m+1}+k-x_{m,i}, q_{i,m+2}-k-1]} = \boldsymbol{E}_{2,[q_{i,m+1}+k, q_{i,m+2}-k-1+x_{\ell,i}]}. \tag{8.30}$$

If no such index or more than one exist, let $x_{m,i} = k+1$. Since $[q_{i,m_\ell}, q_{i,m_\ell+1} - 1] \cap (\boldsymbol{\delta}_1 \cup \gamma_1 \cup \boldsymbol{\delta}_2 \cup \boldsymbol{\delta}_2) \cap \mathcal{I} = \emptyset$, we have that

$$\begin{aligned}
&\boldsymbol{E}_{1,[q_{i,m_\ell}+|\gamma_1 \cap [b_1, q_{i,m_\ell}-1]| - |\boldsymbol{\delta}_1 \cap [b_1, q_{i,m_\ell}-1]|, q_{i,m_\ell+1}-1+|\gamma_1 \cap [b_1, q_{i,m_\ell}-1]| - |\boldsymbol{\delta}_1 \cap [b_1, q_{i,m_\ell}-1]|]} \\
=&\mathbf{c}_{q_{i,m_\ell}, q_{i,m_\ell+1}-1} \\
=&\boldsymbol{E}_{2,[q_{i,m_\ell}+|\gamma_2 \cap [b_1, q_{i,m_\ell}-1]| - |\boldsymbol{\delta}_2 \cap [b_1, q_{i,m_\ell}-1]|, q_{i,m_\ell+1}-1+|\gamma_2 \cap [b_1, q_{i,m_\ell}-1]| - |\boldsymbol{\delta}_2 \cap [b_1, q_{i,m_\ell}-1]|]}
\end{aligned}$$

which implies that the integer $x_{m_\ell,i} \in [-k,k]$ satisfying (8.29) and (8.30) can be found for $\ell \in [1,2k+1]$. According to Proposition 8.6.1, such $x_{m_\ell,i}$ is unique. In the following, we show that

$$|\gamma_w \cap \mathcal{I}| - |\boldsymbol{\delta}_w \cap \mathcal{I}| = \sum_{i:q_{i,m}-1\leq b_2, i\in[1,\frac{b_2-b_1+1}{t}+1]} x_{m_\ell,i} \tag{8.31}$$

for $\ell \in [1,2k+1]$.

For any fixed $\ell \in [1,2k+1]$, let i^* be the largest integer such that $(\gamma_1 \cup \boldsymbol{\delta}_1) \cap \mathcal{I} \cap [1, q_{i,m+1}-1] = \emptyset$. Note that $x_{m_\ell,i} = 0$ for $i \in [1,i^*]$. In addition, we have $|\gamma_w \cap \mathcal{I} \cap [1,q_{i,m+1}-1]| - |\boldsymbol{\delta}_w \cap \mathcal{I} \cap [1,q_{i,m+1}-1| = 0 = \sum_{i=1}^{i^*} x_{m_\ell,i}$ for $w \in \{1,2\}$. According to Proposition 8.6.1 and definition of $x_{m,i}$, we have that

$$\begin{aligned}
x_{m_\ell,i} =& |\gamma_2 \cap [1, q_{i,m+1}+k-1]| - |\boldsymbol{\delta}_2 \cap [1,q_{i,m+1}+k-1]| \\
&- |\gamma_1 \cap [1,q_{i,m+1}+k-1]| + |\boldsymbol{\delta}_\cap [1,q_{i,m+1}+k-1]| \\
\overset{(a)}{=}& |\boldsymbol{\delta}_1 \cap [q_{i-1,m+1}+k, q_{i,m+1}+k-1]| \\
&- |\gamma_1 \cap [q_{i-1,m+1}+k, q_{i,m+1}+k-1]|
\end{aligned}$$

for $i \geq i^*+1$, where (a) follows since $|\gamma_2 \cap [1,q_{i,m+1}+k-1]| = |\gamma_1 \cap [1,q_{i-1,m+1}+k-1]|$ and $|\boldsymbol{\delta}_2 \cap [1,q_{i,m+1}+k-1]| = |\boldsymbol{\delta}_1 \cap [1,q_{i-1,m+1}+k-1]|$. Moreover $x_{m_\ell,i} = 0$ for $q_{i,m_\ell} \geq p_2+1$. Therefore,

$$\begin{aligned}
&\sum_{i:q_{i,m_\ell}-1\leq b_2, i\in[1,\frac{b_2-b_1+1}{t}+1]} x_{m_\ell,i} \\
=& |\boldsymbol{\delta}_1 \cap \mathcal{I} \cap [1,q_{i,m_\ell+1}-1]| - |\gamma_1 \cap \mathcal{I} \cap [1,q_{i,m_\ell+1}-1| \\
&+ \sum_{i:q_{i,m_\ell+1}-1\leq p_2, i\geq i^*+1} (|\boldsymbol{\delta}_1 \cap [q_{i-1,m_\ell+1}, q_{i,m_\ell+1}-1]| \\
&- |\gamma_1 \cap [q_{i-1,m_\ell+1}+k, q_{i,m_\ell+1}+k-1]|) \\
=& |\boldsymbol{\delta}_1 \cap \mathcal{I}| - |\gamma_1 \cap \mathcal{I}|,
\end{aligned}$$

where the last equality holds since interval $\mathcal{I}$ is edit isolated. Therefore, we have (8.31) for $\ell \in [1,2k+1]$. Find the majority of $\sum_{i:q_{i,m}-1\leq b_2, i\in[1,\frac{b_2-b_1+1}{t}+1]} x_{m,i}$ for $m \in [1,4k+1]$, we obtain the value $|\boldsymbol{\delta}_w \cap \mathcal{I}| - |\gamma_w \cap \mathcal{I}|$ for $w \in [1,d]$.

Finally, since $x_{m,i} = 0$ for each pair of (m,i) when $\boldsymbol{E}_{w,[b_1,b_2]} = \boldsymbol{E}_{w',[b_1,b_2]}$ for any $w, w' \in [1,d]$, we have $|\gamma_w \cap \mathcal{I}| = |\boldsymbol{\delta}_w \cap \mathcal{I}|$ for any $w \in [1,d]$. $\square$

The next lemma shows that we can recover most of the bits in $\mathbf{c}$. Before stating the lemma, we define the notion of a minimum edit isolated interval. An interval $\mathcal{I}$ is called a minimum edit isolated interval if:

1. $\mathcal{I}$ cannot be partitioned into two edit isolated intervals, i.e., there do not exists two disjoint edit isolated intervals $\mathcal{I}_a$ and $\mathcal{I}_b$ such that $\mathcal{I} = \mathcal{I}_a \cup \mathcal{I}_b$.
2. There is no strict sub-interval $\mathcal{I}' \subsetneq \mathcal{I}$ of $\mathcal{I}$ that is edit isolated.

We note that the error locations in all heads are caontained in a disjoint set of minimum isolated intervals.

Lemma 8.6.5. *Let $\boldsymbol{E} \in \mathcal{E}_k(\mathbf{c})$ be a read matrix for some sequence $\mathbf{c}$ satisfying $L(\mathbf{c}, \leq k) \leq T$. For an index i not in any minimum edit isolated interval that is disjoint with $\mathcal{I}_j$, $j \in [1, J]$, if the column index of the bit $\boldsymbol{E}_{1,i-|[1:i-1]\cap\delta_1|+|[1:i-1]\cap\gamma_1|}$ coming from c_i in the first read is not contained in one of the output intervals $[b_{1j}, b_{2j}]$, the bit c_i can be correctly recovered given $\boldsymbol{E}$.*

Proof. Assume that $b_{11} < b_{12} < \ldots < b_{1J}$. For each output interval $[b_{1j}, b_{2j}]$, $j \in [1, J]$, let q_j be the largest integer $q_j \in [b_{1j}, b_{2j}]$, such that there exist $w, w' \in [1, d]$ satisfying $\boldsymbol{E}_{w,q_j} \neq \boldsymbol{E}_{w',q_j}$. We show that $q_j \in [b_{1j}+k+1, b_{2j}-k-1]$ for $j \in [1, J]$, unless when $b_{11} = 1$ or $b_{2J} = n' \in [n+1, n+2k+1]$, we can assume that $b_{11} = -k-1$ and $b_{2J} = n+2k+2$, which does not affect the result. Note that for any index i such that there exist $w, w' \in [1, d]$ satisfying $\boldsymbol{E}_{w,i} = \boldsymbol{E}_{w',i}$, the indices $[i+1, i+kdt+t]$ are not marked and contained in some output interval. We have that $q_j \leq b_{2j} - k - 1$. Similarly, $q_j \geq b_{1j} + k + 1$ because the intervals $[q_j - kdt - t, q_j]$, $j \in [1, J]$ is not marked.

According to Lemma 8.6.1, each output interval $[b_{1j}, b_{2j}]$ is associated with an edit isolated interval $\mathcal{I}_j$, $j \in [1, J]$. Moreover, for any index $i \in [n+k+1] \setminus \cup_{j=1}^{J} \mathcal{I}_j$, we have that $\boldsymbol{E}_{w,i-|[1:i-1]\cap\delta_1|+|[1:i-1]\cap\gamma_1|} = \boldsymbol{E}_{w',i-|[1:i-1]\cap\delta_1|+|[1:i-1]\cap\gamma_1|}$ for any $w, w' \in [1, d]$. These imply that for any minimum edit isolated interval $[i_1, i_2]$ that is disjoint with $\mathcal{I}_j$ for $j \in [1, J]$, we have that

$$\begin{aligned}&\boldsymbol{E}_{w,i_1-|[1:i_1-1]\cap\delta_1|+|[1:i_1-1]\cap\gamma_1|,i_2-|[1:i_1-1]\cap\delta_1|+|[1:i_1-1]\cap\gamma_1|}\\ =&\boldsymbol{E}_{w',i_1-|[1:i_1-1]\cap\delta_1|+|[1:i_1-1]\cap\gamma_1|,i_2-|[1:i_1-1]\cap\delta_1|+|[1:i_1-1]\cap\gamma_1|}.\end{aligned} \tag{8.32}$$

By Lemma 8.6.4, we have that $|[i_1, i_2] \cap \delta_1| = |[i_1, i_2] \cap \gamma_1|$, i.e., there is no bit shift caused by errors in interval $[i_1, i_2]$. In addition, the shift caused by errors in interval $\mathcal{I}_j$, $j \in [1, J]$, which is $|\mathcal{I}_j \cap \gamma_1| - |\mathcal{I}_j \cap \boldsymbol{\delta}_1| = s_j$, can be determined. This implies that

$$\begin{aligned} & |[1 : i' - 1] \cap \gamma_1| - |[1 : i' - 1] \cap \boldsymbol{\delta}_1| \\ = & \sum_{j:q_j<i'} s_j \\ \overset{(a)}{=} & \sum_{j:q_j<i'+|[1:i'-1]\cap\gamma_1|+|[1:i'-1]\cap\delta_1|} s_j \end{aligned} \tag{8.33}$$

for any i' satisfying: (1) $i' - |[1 : i' - 1] \cap \boldsymbol{\delta}_1| + |[1 : i' - 1] \cap \gamma_1|$ not in any output interval $[b_{1j}, b_{2j}]$, $j \in [1, J]$. (2) i' is not in any minimum edit isolated interval that is disjoint with $\mathcal{I}_j$, $j \in [1, J]$. The equality (a) holds because $q_j \in [b_{1j} + k + 1, b_{2j} - k - 1]$, and $i' > q_j$ only when $b_{2j} < i' + |[1 : i' - 1] \cap \gamma_1| + |[1 : i' - 1] \cap \boldsymbol{\delta}_1|$. For the same reason, $i' < q_j$ only when $b_{1j} > i' + |[1 : i' - 1] \cap \gamma_1| + |[1 : i' - 1] \cap \boldsymbol{\delta}_1|$.

For any $\boldsymbol{E}_{1,i}$ such that i is not included in any output interval $[b_{1j}, b_{2j}]$, $j \in [1, J]$, let

$$c'_{i-\sum_{j:q_j<i} s_j} = \boldsymbol{E}_{1,i}. \tag{8.34}$$

be an estimate of the bit $c_{i-\sum_{j:q_j<i} s_j}$. Then, for any index i' such that $i' - |[1 : i' - 1] \cap \boldsymbol{\delta}_1| + |[1 : i' - 1] \cap \gamma_1|$ is not included in any output interval and i' is not in any minimum edit isolated interval that is disjoint with $\mathcal{I}_j$, $j \in [1, J]$, we have that

$$\begin{aligned} & c_{i'} \\ = & \boldsymbol{E}_{1,i'-|[1:i'-1]\cap\delta_1|+|[1:i'-1]\cap\gamma_1|} \\ = & c'_{i'-|[1:i'-1]\cap\delta_1-|+|[1:i'-1]\cap\gamma_1|-\sum_{j:q_j<i'-|[1:i'-1]\cap\delta_1-|+|[1:i'-1]\cap\gamma_1|} s_j} \\ = & c'_{i'}, \end{aligned}$$

where the last equality follows from (8.33). Therefore, the proof is done. □

Encoding/Decoding Algorithms

We are now ready to present the encoding and decoding algorithms. We first deal with cases when $d \geq 2d$. The code construction is the similar to the one in Sec. 8.5, stated in Sec. (8.10) and (8.11), which we restate as follows

$$Enc_2(\mathbf{c}) = (F(\mathbf{c}), R_2'(\mathbf{c}), R_2''(\mathbf{c})) \tag{8.35}$$

where

$$R_2'(\mathbf{c}) = RS_{2\lfloor k/d \rfloor}(S(F(\mathbf{c}))),$$

$$R_2''(\mathbf{c}) = Rep_{k+1}(Hash(R_2'(\mathbf{c}))). \tag{8.36}$$

The difference here is in the definition of the function $S(F(\mathbf{c}))$, instead of splitting $F(\mathbf{c})$ into blocks of length B, as in (8.3), we split $F(\mathbf{c})$ into blocks of length $B' = (2kdt+2t+1)(k+1)+kdt+3k$, which is k plus the upper bound on the length of the output intervals $[b_{1j}, b_{2j}]$, $j \in [1, J]$, i.e.,

$$\begin{aligned} F(\mathbf{c}) =& (\mathbf{a}_1', \ldots, \mathbf{a}'_{\lceil \frac{n+k+1}{B'} \rceil}), \text{ and} \\ S(F(\mathbf{c})) =& (Hash(\mathbf{a}_1'), \ldots, Hash(\mathbf{a}'_{\lceil \frac{n+k+1}{B'} \rceil})). \end{aligned} \tag{8.37}$$

It can be verified that the code has the same redundancy $2\lfloor k/d \rfloor \log n + o(\log n)$. In the following, we show that the codeword $Enc_2(\mathbf{c})$ can be correctly decoded.

Similar to what we did in the proof of Theorem 8.3.1 and Theorem 8.5.1, we use the first row in $\boldsymbol{E}$ to decode. From Lemma 8.2.4, we conclude that we can first recover $Hash(R_2'(\mathbf{c}))$ and then $R_2'(\mathbf{c})$ using the deletion correcting hash in Lemma 8.2.3. Apply Lemma 8.6.2 to every output interval $[b_{1j}, b_{2j}]$ and $\boldsymbol{E}_{[1,d],[b_{1j},b_{2j}]}$ and obtain a matrix $\boldsymbol{E}_j'$. If the matrix $\boldsymbol{E}_j'$ cannot be uniquely recovered, let $\boldsymbol{E}_j' = \boldsymbol{E}_{[b_{1j},b_{2j}]}$. By Lemma 8.6.2, the bits $\mathbf{c}_{\mathcal{I}_j}$, where $\mathcal{I}_j$, $j \in [1, J]$ is the corresponding edit isolated interval described in Lemma 8.6.1 for an output interval $[b_{1j}, b_{2j}]$, can be recovered when $|\mathcal{I}_j \cap \gamma_1| + |\mathcal{I}_j \cap \boldsymbol{\delta}_1| \leq d-1$. Furthermore, from Lemma 8.6.4 and Lemma 8.6.5, using (8.34) we can recover the bits $\mathbf{c}_i$ such that i is not in any minimum edit isolated interval and the index $i+|[1,i-1]\cap\gamma_1|-|[1,i-1]\cap\boldsymbol{\delta}_1|$, where $\mathbf{c}_i$ locates in the first row in the read matrix $\boldsymbol{E}$, is not in the output intervals $[b_{1j}, b_{2j}]$. Therefore, we are left to recover the bits $\mathbf{c}_i$ that results in an output interval $[b_{1j}, b_{2j}]$ with at least d errors in the edit isolated interval $\mathcal{I}_j$, and the undetectable minimum edit isolated intervals $[i_1, i_2]$ that are disjoint from $\mathcal{I}_j$, $j \in [1, J]$, which satisfy (8.32). By Lemma 8.6.3, there are at least $2d$ errors in undetectable minimum edit isolated interval $[i_1, i_2]$. Let e be the number of intervals $\mathcal{I}_j$ with at least d errors, and f be the number of undetectable minimum edit isolated intervals. Then we have that $de + 2fd \leq k$ and $e + 2f \leq \lceil \frac{k}{d} \rceil$. Note that a minimum length edit isolated interval has length at most kd. Hence, it covers at most 2 blocks $\mathbf{a}_i'$ and $\mathbf{a}_{i+1}'$ defined in (8.37). In addition, the bits $\mathbf{c}_i$ that results in an output interval $[b_{1j}, b_{2j}]$ with at least d errors in the edit isolated interval $\mathcal{I}_j$ covers at most 2 blocks $\mathbf{a}_{i'}'$ and $\mathbf{a}_{i'+1}'$. These correspond to at most $2e$ erasure errors and $2f$ substitution errors in $S(F(\mathbf{c}))$, since the intervals $[b_{1j}, b_{2j}]$ are known. These errors can be corrected with Reed-Solomon codes correcting $2e + 4f \leq 2\lceil \frac{k}{d} \rceil$ erasure errors. Hence, $S(F(\mathbf{c}))$ can be recovered and then $F(\mathbf{c})$ and $\mathbf{c}$ can be recovered following similar procedures as in Sec. 8.5.

For cases when $d \leq k \leq 2d-1$. The codes are similar to those in Sec. 8.3, which is given by

$$Enc_1(\mathbf{c}) = (F(\mathbf{c}), R_1^{'}(\mathbf{c}), R_1^{''}(\mathbf{c})) \tag{8.38}$$

where

$$\begin{aligned} R_1^{'}(\mathbf{c}) &= ER(S(F(\mathbf{c}))), \\ R_1^{''}(\mathbf{c}) &= Rep_{k+1}(Hash(R_1'(\mathbf{c}))). \end{aligned} \tag{8.39}$$

Similar to cases when $k \geq 2d$, here we split $F(\mathbf{c})$ into blocks of length B', the same as in (8.37). Similar to the redundancy of the code in (8.4). The redundancy is $4k \log\log n + o(\log\log n)$. The correctness of the code is similar to cases when $k \geq 2d$. Note that when $d \leq k \leq 2d-1$, we have that $f = 0$ and $e = 1$, which reduces the decoding to correct 2 consecutive erasures $S(F(\mathbf{c}))$. This can be done using $R_1'(\mathbf{c})$, following similar arguments as in Sec. 8.3. Hence, Theorem 8.1.2 is proved.

8.7 Conclusions

We construct d-head k-deletion racetrack memory codes for any $k \geq d+1$, extending previous works which deals with cases when $k \leq d$. It is proved that for small head distance $t_i = n^{o(1)}$ and for $k \geq 2d$, the redundancy of our codes is asymptotically at most four times the optimal redundancy. We also show the same redundancy results hold for d-head codes correcting a combination of at most k deletions and insertions. Finding a lower bound of the redundancy for $d \leq k \leq 2d-1$ would be interesting, for both deletion correcting codes or codes correcting a combination of deletions and insertions. It is also desirable to tighten the gap between the upper and lower bounds of the redundancy for cases when $k \geq 2d$.

BIBLIOGRAPHY

[1] URL: https://www.idc.com/getdoc.jsp?containerId=prUS47560321.

[2] URL: https://en.wikipedia.org/wiki/Reference_genome.

[3] K. A. Abdel-Ghaffar et al. "On Helberg's generalization of the Levenshtein code for multiple deletion/insertion error correction". In: *IEEE Trans. on Inf. Th* 58.3 (2012), pp. 1804–1808.

[4] F. Ban et al. "Beyond trace reconstruction: Population recovery from the deletion channel". In: *60th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*. pp. 745—768, 2019.

[5] T. Batu et al. "Reconstructing strings from random traces". In: Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). 2004, pp. 910–918.

[6] D. Belazzougui. "Efficient deterministic single round document exchange for edit distance". In: (2015). arXiv: 1511.09229.

[7] D. Belazzougui and Q. Zhang. "Edit distance: Sketching, streaming, and document exchange". In: 2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS). 2016, pp. 51–60.

[8] V. Bhardwaj et al. ""Trace reconstruction problems in computational biology."" In: *IEEE Trans. on Inf. Th* 67.6 (2021), pp. 3295–3314.

[9] P. Borwein. *Computational excursions in analysis and number theory*. Science & Business Media: Springer, 2012.

[10] P. Borwein and T. Erdélyi. ""Littlewood-type problems on subarcs of the unit circle."" In: *Indiana University mathematics journal* 46.4 (1997), pp. 1323–1346.

[11] Peter Borwein, Tamás Erdélyi, and Géza Kós. ""Littlewood-type problems on [0, 1]."" In: *Proceedings of the London Mathematical Society* 79.1 (1999), pp. 22–46.

[12] J. Brakensiek, V. Guruswami, and S. Zbarsky. "Efficient low-redundancy codes for correcting multiple deletions". In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms *(SODA)*. 2016, pp. 1884–1892.

[13] D. Chakraborty, E. Goldenberg, and M. Koucký. "Low distortion embedding from edit to Hamming distance using coupling". In: *Electronic Colloquium on Computational Complexity (ECCC)* 22 (2015), p. 111.

[14] Z. Chang et al. "Rates of DNA string profiles for practical values of read lengths". In: *IEEE Trans. on Inf. Th* 63.11 (2017), pp. 7166–7177.

[15] Z. Chase. “New upper bounds for trace reconstruction”. In: (2020). arXiv: 2009.03296.

[16] Zachary Chase. “New lower bounds for trace reconstruction”. In: *Annales de l’Institut Henri Poincaré, Probabilités et Statistiques*. Vol. 57. 2. Institut Henri Poincaré. 2021, pp. 627–643.

[17] Y. Chee et al. “Codes Correcting limited-shift errors in racetrack memories”. In: *Proc. IEEE Int. Symp. on Inform. Theory*. 2018, pp. 96–100.

[18] Y. Chee et al. “Coding for racetrack memories”. In: *IEEE Trans. Inform. Theory* 64.11 (2018), pp. 7094–7112.

[19] Y. Chee et al. “Reconstruction from deletions in racetrack memories”. In: *Proc. IEEE Inform. Theory Workshop*. 2018.

[20] X. Chen et al. “Polynomial-time trace reconstruction in the low deletion rate regime”. In: (2020). arXiv: 2012.02844.

[21] X. Chen et al. “Polynomial-time trace reconstruction in the smoothed complexity model”. In: Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). 2021, pp. 54–73.

[22] K. Cheng et al. “Deterministic document exchange protocols, and almost optimal binary codes for edit errors”. In: IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS). 2018, pp. 200–211.

[23] M. Cheraghchi et al. “Coded trace reconstruction”. In: *IEEE Trans. Info. Th* 66.10 (2020), pp. 6084–6103.

[24] G. M. Church, Y. Gao, and S. Kosuri. “Next-generation digital information storage in DNA”. In: *Science* 6102 (2012), pp. 1628–1628.

[25] G. Cormode et al. “Communication complexity of document exchange”. In: Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). 2000, pp. 197–206.

[26] S. Davies et al. “Approximate trace reconstruction”. In: *Proc. IEEE Int. Symp. on Inform. Theory*. 2021.

[27] A. De, R. O’Donnell, and R. A. Servedio. “Optimal mean-based algorithms for trace reconstruction”. In: *The Annals of Applied Probability* 29.2 (2019), pp. 851–874.

[28] O. Elishco et al. “Repeat-free codes”. In: *IEEE Trans. on Inf. Th* 67.9 (2021), pp. 6084–6103.

[29] Y. Erlich and D. Zielinski. “DNA fountain enables a robust and efficient storage architecture”. In: *Science* 6328 (2017), pp. 950–954.

[30] A. Extance. “How DNA could store all the world’s data”. In: *Nature* 537 (2016), pages22–24.

[31] R. P. Feynman. “There’s plenty of room at the bottom: An invitation to enter a new field of physics”. In: *Handbook of Nanoscience, Engineering, and Technology,Third Edition, CRC Press* (2012), pp. 26–35.

[32] R. Gabrys, H. M. Kiah, and O. Milenkovic. “Asymmetric Lee distance codes for DNA-based storage”. In: *IEEE Trans. on Inf. Th* 63.8 (2017), pp. 4982–4995.

[33] R. Gabrys and O. Milenkovic. “Unique reconstruction of coded strings from multiset substring spectra”. In: *IEEE Trans. Inf. Th* 65.12 (2019), pp. 7682–7696.

[34] R. Gabrys and F. Sala. “Codes correcting two deletions”. In: *IEEE Trans. on Inf. Th* 65.2 (Feb. 2019), pp. 965–974.

[35] R. Gabrys, E. Yaakobi, and O. Milenkovic. “Codes in the Damerau Distance for Deletion and Adjacent Transposition Correction”. In: *IEEE Trans. on Inf. Th* 64.4 (2018), pp. 2550–2570.

[36] N. Goldman et al. “Towards practical, high-capacity, low-maintenance information storage in synthesized DNA”. In: *Nature* 7435 (2013), pp. 77–80.

[37] E. Grigorescu, M. Sudan, and M. Zhu. “Limitations of Mean-Based Algorithms for Trace Reconstruction at Small Edit Distance”. In: *IEEE Transactions on Information Theory* (2022).

[38] V. Guruswami and J. Håstad. “Explicit Two-Deletion Codes With Redundancy Matching the Existential Bound”. In: *IEEE Trans. Inf. Theory* 67.10 (2020), pp. 6384–6394.

[39] V. Guruswami, A. Rudra, and M. Sudan. *Essential coding theory*. Draft available at https://cse.buffalo.edu/faculty/atri/courses/coding-theory/book/web-coding-book.pdf, 2022.

[40] V. Guruswami and C. Wang. “Deletion codes in the high-noise and high-rate regimes”. In: *IEEE Trans. on Inf. Th* 63.4 (2017), pp. 1961–1970.

[41] B. Haeupler. “Optimal document exchange and new codes for small number of insertions and deletions”. In: IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS). 2019, pp. 334–347.

[42] B. Haeupler and ”Synchronization strings A. Shahrasbi. “codes for insertions and deletions approaching the singleton bound.”” In: *ACM SIGACT Symposium on Theory of Computing (STOC)*. 2017, pp. 33–46.

[43] M. Hagiwara. *On ordered syndromes for multi insertion/deletion error-correcting codes*. 2016.

[44] S. K. Hanna and S. El Rouayheb. “Guess & check codes for deletions, insertions, and synchronization”. In: *IEEE Trans. on Inf. Th* 65.1 (Jan. 2019), pp. 3–15.

[45] M. Hayashi et al. “Current-controlled magnetic domain-wall nanowire shift register”. In: *Science* 320.5873 (2008), pp. 209–211.

[46] R. Heckel et al. “Fundamental limits of DNA storage systems”. In: Proc. IEEE Int. Symp. Inf. Theory *(ISIT*. 2017, pp. 3130–3134.

[47] A. S. Helberg and H. C. Ferreira. “On multiple insertion/deletion correcting codes”. In: *IEEE Trans. on Inf. Th* 48.1 (2002), pp. 305–308.

[48] N. Holden, R. Pemantle, and Y. Peres. “Subpolynomial trace reconstruction for random strings and arbitrary deletion probability”. In: *Proceedings of the 31st Conference On Learning Theory (COLT)* (2018), pp. 1799–1840.

[49] T. Holenstein et al. “Trace reconstruction with constant deletion probability and related results”. In: *Proc. 19th ACM-SIAM Symposium on Discrete Algorithms (SODA)*. pp. 389—398, 2008.

[50] U. Irmak, S. Mihaylov, and T. Suel. “Improved single-round protocols for remote file synchronization”. In: *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)* 3 (2005), pp. 1665–1676.

[51] H. Jowhari. “Efficient communication protocols for deciding edit distance”. In: European Symposium on Algorithms. 2012, pp. 648–658.

[52] H. M. Kiah, G. J. Puleo, and O. Milenkovic. “Codes for DNA sequence profiles”. In: Proc. IEEE Int. Symp. Inf. Theory *(ISIT*. 2015, pp. 814–818.

[53] D. E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 3: Generating All Combinations and Partitions (Art of Computer Programming),* Addison-Wesley Professional, 2005.

[54] M. Kovačević and V. Y. F Tan. “Codes in the space of multisets–Coding for permutation channels with impairments”. In: *IEEE Trans. on Inf. Th* 64.7 (2018), pp. 5156–5169.

[55] M. Kovačević and D. Vukobratović. “Perfect codes in the discrete simplex”. In: *Designs, Codes and Cryptography* 75.1 (2015), pp. 81–95.

[56] I. Krasikov and Y. Roditty. “On a reconstruction problem for sequences”. In: *Journal of Combinatorial Theory, Series A* 77.2 (1997), pp. 344–348.

[57] Akshay Krishnamurthy et al. “Trace reconstruction: Generalized and parameterized”. In: *IEEE Transactions on Information Theory* 67.6 (2021), pp. 3233–3250.

[58] M. Langberg, M. Schwartz, and E. Yaakobi. “Coding for the ℓ_∞-limited permutation channel”. In: *IEEE Trans. on Inf. Th* 63.12 (2017), pp. 7676–7686.

[59] T. A. Le and H. D. Nguyen. “New multiple insertion/deletion correcting codes for non-binary alphabets”. In: *IEEE Trans. on Inf. Th* 62.5 (2016), pp. 2682–2693.

[60] A. Lenz and N. Polyanskii. “Optimal codes correcting a burst of deletions of variable length”. In: *Proc. IEEE Int. Symp. Inf. Theory*. USA: Los Angeles, 2020.

[61] A. Lenz et al. “Anchor-based correction of substitutions in indexed sets”. In: *Proc. IEEE Int. Symp. Inf. Theory*. Paris, France: ISIT), 2019.

[62] A. Lenz et al. “Coding over sets for DNA storage”. In: *IEEE Trans. on Inf. Th* 66.4 (2020), pp. 2331–2351.

[63] V. I. Levenshtein. “Asymptotically optimal binary code correcting loss of one or two adjacent bits”. In: *(in Russian), Probl. Cybern* (1967), pp. 293–298.

[64] V. I. Levenshtein. “Binary codes capable of correcting deletions, insertions, and reversals”. In: *Soviet physics doklady* 10.8 (1966), pp. 707–710.

[65] V. I. Levenshtein. “Bounds for deletion/insertion correcting codes”. In: *Proc. IEEE Int. Symp. Inf. Theory*. 2002.

[66] V. I. Levenshtein. “Efficient Reconstruction of Sequences”. In: *IEEE Trans. on Inf. Th* 47.1 (2001), pp. 2–22.

[67] V. I. Levenshtein. “Reconstructing objects from a minimal number of distorted patterns”. In: *(in Russian),* Dokl. Acad. Nauk 354 (), pp. 593–596.

[68] M. Luby. “LT codes”. In: The 43rd Annual IEEE Symposium on Foundations of Computer Science *(FOCS*. 2002.

[69] M. Mitzenmacher. “A survey of results for deletion channels and related synchronization channels”. In: *Probability Surveys* 6 (2009), pp. 1–33.

[70] F. Nazarov and Y. Peres. “Trace reconstruction with $\exp(O(n^{1/3}))$ samples”. In: Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC). 2017, pp. 1042–1046.

[71] J. L. Nicolas. “On highly composite numbers”. In: Ramanujan revisited, *Urbana-Champaign, Ill.* 1987, pp. 215–244.

[72] R. O’Donnell. *Analysis of boolean functions*. Cambridge University Press, 2014.

[73] L. Organick et al. “Scaling up DNA data storage and random access retrieval”. In: (2017).

[74] F. Paluncic et al. “A multiple insertion/deletion correcting code for run-length limited sequences”. In: *IEEE Trans. on Inf. Th* 58.3 (2012), pp. 1809–1824.

[75] S. S. Parkin, M. Hayashi, and L. Thomas. “Magnetic domain-wall racetrack memory”. In: *Science* 320.5873 (2008), pp. 190–194.

[76] Y. Peres and A. Zhai. “Average-case reconstruction for the deletion channel: Subpolynomially many traces suffice”. In: 58th IEEE Annual Symposium on Foundations of Computer Science (FOCS). 2017, pp. 228–239.

[77] N. Raviv, M. Schwartz, and E. Yaakobi. “Rank-modulation codes for DNA storage with shotgun sequencing”. In: *IEEE Trans. on Inf. Th* 65.1 (2019), pp. 50–64.

[78] J. M. Robson. “Separating strings with small automata”. In: *Information Processing Letters* 30.4 (1989), pp. 209–214.

[79] R. Roth. *Introduction to coding theory, Cambridge University Press*. 2006.

[80] C. Schoeny et al. “Codes Correcting a Burst of Deletions or Insertions”. In: *IEEE Trans. on Inf. Th* 63.4 (2017), pp. 1971–1985.

[81] L. J. Schulman and D. Zuckerman. “Asymptotically good codes correcting insertions, deletions, and transpositions”. In: *IEEE Trans. on Inf. Th* 45.7 (1999), pp. 2552–2557.

[82] F. Sellers. “Bit loss and gain correction codes”. In: *IRE Transactions on Information Theory* 8.1 (1962), pp. 35–38.

[83] T. Shinkar et al. “Clustering-correcting codes”. In: *Proc. IEEE Int. Symp. Inf. Theory*. Paris, France: ISIT), 2019.

[84] A. Shokrollahi. “Raptor codes”. In: *IEEE/ACM Transactions on Networking* 14 (2006), pp. 2551–2567.

[85] I. Shomorony and R. Heckel. “Capacity results for the noisy shuffling channel”. In: *Proc. IEEE Int. Symp. Inf. Theory*. Paris, France: ISIT), 2019.

[86] J. Sima and J. Bruck. “On Optimal k-Deletion Correcting Codes”. In: *IEEE Trans. on Inf. Th* 67.6 (2020), pp. 3360–3375. DOI: 10.1109/TIT.2020.3028702.

[87] J. Sima, R. Gabrys, and J. Bruck. “Optimal Systematic t-Deletion Correcting Codes”. In: *Proc. IEEE Int. Symp. Inf. Theory*. Los Angeles, CA, USA: IEEE, 2020. DOI: 10.1109/ISIT44484.2020.9173986.

[88] J. Sima, N. Raviv, and J. Bruck. “Robust Indexing - Optimal Codes for DNA Storage”. In: *Proc. IEEE Int. Symp. Inf. Theory*. Los Angeles, CA, USA: IEEE, 2020. DOI: 10.1109/ISIT44484.2020.9174447.

[89] W. Song, K. Cai, and K. A. S. Immink. “Sequence-subset distance and coding for error control for DNA-based data storage”. In: *Proc. IEEE Int. Symp. Inf. Theory*. Paris, France: ISIT), 2019.

[90] W. Song et al. “On Multiple-Deletion Multiple-Substitution Correcting Codes”. In: *Proc. IEEE Int. Symp. on Inform. Theory*. 2021, pp. 2655–2660.

[91] Z. Sun, W. Wu, and H. Li. “Cross-layer racetrack memory design for ultra high density and low power consumption”. In: *Design Automation Conference (DAC), 50 th ACM/EDAC/IEEE*. 2013, pp. 1–6.

[92] I. Tal et al. “Polar codes for the deletion channel: Weak and strong polarization”. In: *Proc. IEEE Int. Symp. Inf. Theory*. Paris, France, 2019.

[93] G. Tenengolts. “Nonbinary codes, correcting single deletion or insertion”. In: *IEEE Trans. Inform. Theory* 30.5 (1984), pp. 766–769.

[94] A. Vahid et al. “”Correcting Two Deletions and Insertions in Racetrack Memory,” in”. *arXiv preprint*, archivePrefix = arXiv, eprint = 1701.06478. 2017.

[95] R. R. Varshamov and G. M. Tenengolts. “Codes which correct single asymmetric errors”. In: *Autom. Remote Control* 26.2 (1965), pp. 286–290.

[96] J. M. Walsh and S. Weber. “Capacity region of the permutation channel”. In: *46th Annual Allerton Conference on Communication, Control, and Computing*. 2008, pp. 646–652.

[97] L. R. Welch and E. R. Berlekamp. “Error correction for algebraic block codes”. In: *US Patent Number 4,633,470* (Dec. 1986).

[98] S. Yang, C. Schoeny, and L. Dolecek. “Theoretical bounds and constructions of codes in the generalized cayley metric”. In: *IEEE Trans. on Inf. Th* 65 (2019), p. 8.

[99] S. M. H. T. Yazdi et al. “A rewritable, random-access DNA-based storage system”. In: *Scientific reports* 5.14138 (2015).

[100] C. Zhang et al. “Hi-fi playback: Tolerating position errors in shift operations of racetrack memory”. In: *ACM SIGARCH Computer Architecture News*. 2015, pp. 694–706.