# GUESS \& CHECK CODES FOR DELETIONS, INSERTIONS, AND SYNCHRONIZATION 

by<br>SERGE KAS HANNA

A dissertation submitted to the<br>School of Graduate Studies<br>Rutgers, The State University of New Jersey<br>In partial fulfillment of the requirements<br>For the degree of Doctor of Philosophy Graduate Program in Electrical and Computer Engineering<br>Written under the direction of Salim El Rouayheb<br>And approved by

$\qquad$
$\qquad$
$\qquad$
$\qquad$
$\qquad$

New Brunswick, New Jersey
October, 2020

# ABSTRACT OF THE DISSERTATION 

# Guess \& Check Codes for Deletions, Insertions, and Synchronization 

## By SERGE KAS HANNA

## Dissertation Director:

## Salim El Rouayheb

Deletion and insertion errors are experienced in various communication and storage systems. These errors are often associated with various forms of loss of synchronization. Constructing codes for correcting deletions and insertions with optimal redundancy, and efficient encoding and decoding, has been and remains in general an open problem. The study of deletion correcting codes has several applications, such as file synchronization and DNA-based storage. This thesis will study the problem of designing codes for deletions, insertions, and synchronization, and will be divided into three parts.

In the first part, this thesis focuses on the problem of constructing codes that can correct $\delta$ deletions that are arbitrarily located in a binary string. The fundamental limits for this problem, derived by Levenshtein, show that the optimal number of redundant bits needed to correct $\delta$ deletions in an $n$-bit codeword is $\mathcal{O}(\delta \log n)$. Varshamov-Tenengolts (VT) codes, dating back to 1965 , are zero-error single deletion $(\delta=1)$ correcting codes, that have an asymptotically
optimal redundancy of value at most $\log (n+1)$ bits, and have linear time $\mathcal{O}(n)$ encoding and decoding algorithms. Finding similar codes for $\delta \geq 2$ deletions remains an open problem. Classical deletion correcting codes require correcting all $\delta$ deletions with zero-error. In our work, we relax the standard zero-error decoding requirement, and instead require correcting almost all $\delta$ deletions (a fraction that goes to 1 with $n$ ). One of our main contributions is an explicit construction of a new family of codes, that we call Guess \& Check (GC) codes, that can correct with high probability up to a constant number of $\delta$ deletions (or insertions). GC codes are systematic and have $c(\delta+1) \log k$ redundancy, where $k$ is the length of the information message and $c>\delta$ is a code parameter. Moreover, these codes have deterministic encoding and decoding algorithms that run in polynomial time in $k$. GC codes are, so far, the only existing systematic codes with logarithmic redundancy, which makes these codes suitable for file synchronization applications. We describe the application of GC codes to file synchronization, and highlight the resulting savings in terms of communication cost and number of communication rounds.

In the second part of this thesis, we study the problem of correcting deletions that are localized in certain parts of the codeword, which are unknown a priori. This study is motivated by file synchronization, in applications such as cloud storage, where large files are often edited by deleting and inserting characters in a relatively small part of the file (such as editing a paragraph). The model we study is when $\delta$ deletions are localized in a window of size $w$ in the codeword. This model is a generalization of the bursty model in which all deletions must be consecutive. Our main contribution in this part is constructing new explicit codes for the localized model, that can correct, with high probability, $\delta \leq w$ deletions that are localized in a single window of size $w$, where $w=o(k)$ grows as a sublinear function of the length of the information message $k$. Therefore, we extend existing results in the literature which study the problem for a window size that is fixed to $w=3$ or $w=4$. Furthermore, the encoding complexity of our codes is $o\left(k^{2}\right)$, and the decoding complexity is $\mathcal{O}\left(k^{2}\right)$.

In the third part of this thesis, we study the problem of coded trace reconstruction which has applications to DNA-based storage. DNA-based storage systems introduce various challenges,
such as errors in the stored data due to DNA breakages caused by chemical reactions, or errors resulting from the process of DNA sequencing. For instance, DNA sequencing using nanopores results in multiple traces (copies) of the data which contain errors such as deletions. One solution to enhance the reliability of such systems is to code the stored data. In coded trace reconstruction, the goal is to design codes that allow reconstructing the data efficiently from a small a number of traces. We study the model where the traces are obtained as outputs of independent deletion channels, where each channel deletes each bit of the input codeword $x$ independently with probability $p$. Our main contribution in this part is designing novel codes for trace reconstruction with constant redundancy, which allow reconstructing a coded sequence from a constant number of traces, in the regime where $p=\Theta(1 / n)$. Our results improve on the state of the art coded trace reconstruction algorithm, which requires a logarithmic redundancy in $n$ for a similar regime where the number of deletions in each trace is fixed.

## Acknowledgements

I owe my deepest gratitude to my mother Marlene Mardikian. Thank you for your countless sacrifices and support, and for setting the bar so high. Without you this work would never have been possible. I am also forever grateful to my grandmother Sona and late grandfather Levon Mardikian for their unconditional love throughout my upbringing.

I wish to express my gratitude to my advisor Dr. Salim El Rouayheb, for his guidance and continuous support. Thank you for the questions and ideas you brought up during our discussions. Your intuitive approach to research in general, helped me improve my papers and presentations, and also my understanding of my research problems. I also appreciate your flexibility and trust, which created a very comfortable work environment. I was lucky to have two of the most experienced professors and researchers in my field of study on my comprehensive exam committee, thesis proposal committee, and thesis defense committee. Dr. Emina Soljanin and Dr. Roy Yates, thank you for your time and for your constructive feedback. I also want to thank Dr. Antonia Wachter-Zeh for being flexible and agreeing to attend my thesis defense from Germany despite the time difference. Further, I thank Dr. Camilla Hollanti and Dr. Rafael Schaefer for hosting me during my summer research visits to Aalto University, Finland and Technical University of Berlin, Germany.

I was fortunate to have supporting friends and companions by my side throughout my journey. Samara Gharbieh, thank you for your love and support and for being there for me through thick and thin. Your presence by my side helped me fulfill this chapter of my life although you were physically thousands of miles away. I also would like to thank my housemate,
labmate, and dear friend Rawad Bitar. You were an amazing companion, and I cherish all the moments spent together in Chicago, New Jersey, Finland, and Germany. During my doctoral studies, I shared my office with several friends: Razane Tajjedine, Peiwen Tian, Carolina Naim, Ghadir Ayache, Rafael D'Oliveira, and Fangwei Ye. I want to thank all of them for making our office an enjoyable workplace. Elie Bahouth, Charbel Akoury, Elie Allaf, Paul Dirany, and Elias Khoueiry, thank you for all the memories and for making my vacation trips to Lebanon so refreshing.

## Dedication

To my mother Marlene, grandmother Sona, and late grandfather Levon.

## Table of Contents

Abstract ..... ii
Acknowledgements ..... v
Dedication ..... vii
List of Figures ..... xii
List of Tables ..... xv

1. Introduction ..... 1
1.1. Motivation ..... 1
1.2. Guess \& Check Codes for Unrestricted Deletions ..... 2
1.2.1. Related Work on Deletions and Synchronization ..... 2
1.2.2. Contributions of Chapter 2 ..... 4
1.2.3. Application of Guess \& Check Codes to File Synchronization ..... 10
1.3. Codes for Localized Deletions ..... 11
1.3.1. Related Work on Bursts of Deletions ..... 11
1.3.2. Contributions of Chapter 3 ..... 12
1.4. Codes for Trace Reconstruction ..... 13
1.4.1. Related Work on Trace Reconstruction ..... 14
1.4.2. Contributions of Chapter 4 ..... 16
1.5. Dissertation Outline ..... 16
2. Guess \& Check Codes for Unrestricted Deletions ..... 17
2.1. Introduction ..... 17
2.2. Notation ..... 19
2.3. Main Result ..... 20
2.4. Encoding and Decoding Examples ..... 21
2.5. General Encoding and Decoding of GC Codes ..... 26
2.5.1. Encoding Steps ..... 27
2.5.2. Decoding Steps ..... 27
2.6. Proof of Theorem 2.1 ..... 28
2.6.1. Redundancy ..... 28
2.6.2. Complexity ..... 29
2.6.3. Proof of the probability of decoding failure for $\delta=1$ ..... 30
2.6.4. Proof of the probability of decoding failure for $\delta$ deletions ..... 33
2.6.5. Proof of Claim 2.5 ..... 34
2.7. Trade-offs ..... 39
2.8. Correcting $\delta$ Insertions ..... 40
2.9. Simulation Results ..... 41
2.10. Application to File Synchronization ..... 42
2.11. List Decoding ..... 43
2.11.1. Preliminaries ..... 45
2.11.2. Theoretical Results ..... 47
2.11.3. Numerical Results ..... 49
2.12. Conclusion ..... 51
3. Codes for Localized Deletions ..... 53
3.1. Introduction ..... 53
3.2. Preliminaries ..... 55
3.3. Main Results ..... 57
3.4. Encoding and Decoding Examples ..... 60
3.5. Encoding and Decoding for Localized Deletions ..... 64
3.5.1. Encoding using GC codes ..... 65
3.5.2. Decoding using GC codes ..... 66
3.5.3. Discussion ..... 67
3.6. Simulation Results ..... 69
3.6.1. Results for $c=3 \mathrm{MDS}$ parity symbols ..... 69
3.6.2. Results for $c=4$ and $c=5$ MDS parity symbols ..... 70
3.6.3. Comparison to the theoretical upper bound in Theorem 3.2 ..... 70
3.7. Correcting Deletions Localized in Multiple Windows ..... 71
3.8. Proof of Theorem 3.1 and Theorem 3.2 ..... 73
3.8.1. Redundancy ..... 73
3.8.2. Complexity ..... 73
3.8.3. Probability of Decoding Failure ..... 74
3.8.4. Proof of Claim 3.12 ..... 77
3.9. Conclusion ..... 80
4. Codes for Trace Reconstruction ..... 81
4.1. Introduction ..... 81
4.2. Preliminaries ..... 82
4.3. Code Construction ..... 83
4.3.1. Overview ..... 83
4.3.2. Bitwise Majority Alignment ..... 84
4.3.3. Delimiter Bits ..... 85
4.3.4. Code Construction ..... 88
4.3.5. Discussion ..... 89
4.4. Results ..... 91
4.4.1. Theoretical Results ..... 91
4.4.2. Simulation Results ..... 95
4.5. Conclusion ..... 98
5. Conclusion ..... 99
References ..... 102
Appendices ..... 107
Appendix A. Additional Proofs for Chapter 2 ..... 108
A.1. Proof of Claim 2.6 ..... 108
A.2. Proof of Theorem 2.7 ..... 111
Appendix B. Additional Proofs for Chapter 3 ..... 113
B.1. Proof of Theorem 3.8 and Theorem 3.9 ..... 113
Appendix C. Additional Proofs for Chapter 4 ..... 115
C.1. Proof of Lemma 4.6 ..... 115

## List of Figures

1.1. The results are averaged over 1000 runs. In each run, a string of size 1 Mb is chosen uniformly at random, and the file to be synchronized is obtained by deleting $d$ bits from it uniformly at random. The total communication cost is expressed in Kb .
2.1. General encoding block diagram of GC codes for $\delta$ deletions. Block I: The binary message of length $k$ bits is chunked into adjacent blocks of length $\log k$ bits each, and each block is mapped to its corresponding symbol in $G F(q)$ where $q=2^{\log k}=k$. Block II: The resulting string is coded using a systematic $(k / \log k+c, k / \log k) q$-ary MDS code where $c>\delta$ is the number of parity symbols. Block III: The symbols in $G F(q)$ are mapped to their binary representations. Block IV: Only the parity bits are coded using a $(\delta+1)$ repetition code.
2.2. The figure shows the maximum list size $L_{\max }$ obtained by GC codes and the codes in [1] for $\delta=2$ deletions and different message lengths. The GC code parameters are set to $c=\delta+1$ and $\ell=\log k$. The results are obtained over 10000 runs of simulations. In each run, the message and the deletion pattern are chosen independently and uniformly at random.
3.1. Encoding block diagram of GC codes for correcting $\delta \leq w$ deletions that are localized within a single window of size at most $w$ bits. Block I: The binary message of length $k$ bits is chunked into adjacent blocks of length $\ell$ bits each, and each block is mapped to its corresponding symbol in $G F(q)$ where $q=2^{\ell}$. Block II: The resulting string is encoded using a systematic $(k / \ell+c, k / \ell) q$-ary MDS code where $c$ is the number of parity symbols and $q=k>k / \ell+c$. Block III: The symbols in $G F(q)$ are mapped to their binary representations. Block IV: A buffer of $w$ zeros followed by a single one is inserted between the systematic and the parity bits.
3.2. The figures show the code rate $R=k / n$ and the empirical probability of decoding failure $\operatorname{Pr}(F)$ of GC codes for different message lengths $k$ and number of deletions $\delta$. The $\delta$ deletions are localized within a window of size $w=\log k$ bits. The values of $R$ are rounded to the nearest $10^{-2}$. The results of $\operatorname{Pr}(F)$ are averaged over $10^{5}$ runs of simulations. In each run, a message $\mathbf{u}$ chosen uniformly at random is encoded into the codeword $\mathbf{x}$. The positions of the window and the deletions in $\mathbf{x}$ are correspondingly chosen uniformly at random. The resulting string is then decoded
3.3. The values of the code rate $R$ and the empirical probability of failure $\operatorname{Pr}(F)$ for different message lengths $k$ and number of localized deletions $\delta$, for the same experimental setup described in Fig. 3.2. The table also shows that GC codes perform better than what the theoretical upper bound in Theorem 3.2 indicates.
4.1. The table shows the observations in positions $\ell-1$ and $\ell$, at the decoder if the first block of $x \in \mathcal{D}_{1}$ is affected by at most one deletion after it passes through the deletion channel. The corresponding decision rules are also given. . . . . . . . 87
4.2. The table shows the observations in positions $\ell-2, \ell-1 \ell$, at the decoder if the first block if $x \in \mathcal{D}_{2}$ is affected by at most two deletions after it passes through the deletion channel. $\times$ refers to an irrelevant entry. The corresponding decision rules are also given.
4.3. Comparing the probability of errors for the uncoded and coded cases $P_{\text {uncoded }}$ and $P_{\text {coded }}$, for $n=1000, k_{1}=3, k_{2}=5$ and $\ell=200$.
4.4. Empirical probability of error of the codes $\mathcal{C}_{1}$ and $\mathcal{C}_{2}$ for multiple values of the number of traces $t$. The length of the codeword is $n=10^{4}$ bits, and the code parameters are $k_{2}=10$ and $k_{1}=5$. The results are averaged over $10^{4}$ trials.
4.5. Empirical probability of error of the codes $\mathcal{C}_{1}$ and $\mathcal{C}_{2}$ for multiple values of the length of the codeword $n$. The length of the codeword is $t=4$ bits, and the code parameters are $k_{2}=10$ and $k_{1}=5$. The results are averaged over $10^{4}$ trials.
4.6. Empirical probability of error of the code $\mathcal{C}_{1}$ with redundancy $4 k_{2}$, for multiple values of the code parameter $k_{2}$. The length of the codeword is $n=10^{4}$ bits, the number of traces is $t=6$, and the deletion probability is $p=k_{1} / n=5 / n$. The results are averaged over $10^{4}$ trials.
4.7. Empirical probability of error of the code $\mathcal{C}_{1}$ with redundancy $4 k_{2}$, for multiple values of the code parameter $k_{2}$. The length of the codeword is $n=10^{4}$ bits, the number of traces is $t=4$, and the deletion probability is $p=k_{1} / n=5 / n$. The results are averaged over $10^{4}$ trials.

## List of Tables

2.1. The table shows the code rate $R=k / n$ and the probability of decoding failure $\operatorname{Pr}(F)$ of GC codes for different message lengths $k$ and different number of deletions $\delta$. The results shown are for $c=\delta+1$ and $\ell=\log k$. The results of $\operatorname{Pr}(F)$ are averaged over 10000 runs of simulations. In each run, a message $\mathbf{u}$ chosen uniformly at random is encoded into the codeword $\mathbf{x} . \delta$ bits are then deleted uniformly at random from $\mathbf{x}$, and the resulting string is decoded.
2.2. Results are averaged over 1000 runs. In each run, a string of size 1 Mb is chosen uniformly at random, and the file to be synchronized is obtained by deleting $d$ bits from it uniformly at random. The total communication cost is expressed in bits. The number of center bits used is 25 .
2.3. The table shows the average list size $L_{a v}$ and the maximum list size $L_{\text {max }}$ obtained by the GC decoder for different message lengths $k$ and different number of deletions $\delta$. The results shown are for $c=\delta+1$ and $\ell=\log k$. The results of $L_{a v}$ and $L_{\max }$ were recorded over 10000 runs of simulations. In each run, a message $\mathbf{u}$ chosen uniformly at random is encoded into the codeword $\mathbf{x} . \delta$ bits are then deleted from $\mathbf{x}$ based on a uniformly distributed deletion pattern $\mathbf{d}$, and the resulting string is decoded.

[^0]
## Chapter 1

## Introduction

### 1.1 Motivation

Deletions and insertions are experienced in various communication and storage systems. In communication, bits that are deleted in a transmitted sequence, are completely removed from this sequence, and their locations are unknown at the receiver. For example, if 1010 is transmitted, the receiver would get 00 if the first and third bits were deleted. In error control coding, dealing with these errors is more difficult than other types of errors such as substitutions (bit flips) and erasures. Deletion and insertion errors are often associated with various forms of loss of synchronization [1-4]. For instance, in file synchronization applications such as cloud storage, the file edits of a user are a result of a series of deletions, insertions, and substitutions of characters. In that setting, when a user edits a file at one end, the goal is to synchronize the versions of other users with a minimum number of communicated bits $[1-3,5,6]$. Another application, where such errors are experienced, is DNA-based storage [7]. In such systems, media degradation arises due to DNA aging caused by chemical reactions (humidity, temperature). As an example, human cellular DNA undergoes anywhere between 10-50 breakages in a cell cycle [7]. These DNA breakages lead to data symbol deletions. Furthermore, the process of DNA sequencing results in multiple erroneous copies of the data, where the errors contain
deletions and insertions [8]. This motivates the study of deletion and insertion correcting codes in this dissertation, and their application to various communication and storage systems.

### 1.2 Guess \& Check Codes for Unrestricted Deletions

In this section, we summarize our results in Chapter 2 on constructing binary codes that can correct multiple unrestricted deletions (or insertions). By unrestricted we mean that the deletions can occur at arbitrary positions in the codeword, which would be in contrast to other models that we will describe later. Namely, consider a binary string of length $n$ bits that is affected by up to $\delta$ deletions, resulting in a shorter string of length at least $n-\delta$. The goal is to design codes that can efficiently correct the $\delta$ deletions, where we focus on the following code properties: redundancy; encoding and decoding complexity; and probability of error.

In this dissertation, we are particularly motivated by the application of deletion correcting codes to remote file synchronization. Consider the setting where two remote nodes (servers) are connected by a noiseless communication link and have different copies of the same file, where one copy is an edited version of the other. The edits in general include deletions and insertions of characters. The goal is to synchronize the two files with minimal number of communicated bits. For the sake of simplicity, let us assume the edits consist of deletions only. If a deletion correcting code is systematic, then it can be used to synchronize the files in the following manner. One node uses the code to compute the parity bits of its file and transfers these bits to the other node. Then, the other node uses these parity bits to decode the deletions. This shows the utility of systematic deletion correcting codes in file synchronization applications.

### 1.2.1 Related Work on Deletions and Synchronization

The deletion channel is probably the most notorious example of a point-to-point channel whose capacity remains unknown. The capacity of the deletion channel has been studied in the probabilistic model. In the model where the deletions are i.i.d. and occur with a fixed probability
$p$, an immediate upper bound on the channel capacity is given by the capacity of the erasure channel $1-p$. Mitzenmacher and Drinea showed in [9] that the capacity of the deletion channel in the iid model is at least $(1-p) / 9$. Extensive work in the literature has focused on determining lower and upper bounds on the capacity [9-15]. We refer interested readers to the comprehensive survey by Mitzenmacher [16]. Ma et al. [3] also studied the capacity in the bursty model of the deletion channel, where the deletion process is modeled by a Markov chain.

A separate line of work has focused on constructing zero-error codes that can correct a fixed number of deletions $\delta$ that are arbitrarily located in a binary string. Levenshtein showed in [17] that the Varshamov-Tenengolts (VT) codes [18] are capable of correcting a single deletion $(\delta=1)$ with zero-error. The redundancy of VT codes is asymptotically optimal $(\log (n+1)$ bits). Abdel-Ghaffar et al. introduced a systematic encoding algorithm for these codes [19]. More information about VT codes and other properties of single deletion codes can be found in [20]. VT codes have also been used to construct codes that can correct a combination of a single deletion and multiple adjacent transpositions [7]. However, finding VT-like codes for multiple deletions $(\delta \geq 2)$ is an open problem. In [17], Levenshtein provided bounds showing that the asymptotic number of redundant bits needed to correct $\delta$ bit deletions in an $n$ bit codeword is $c \delta \log n$ for some constant $c>0$. Levenshtein's bounds were later generalized and improved in [21]. The simplest zero-error code for correcting $\delta$ deletions is the $(\delta+1)$ repetition code, where every bit is repeated $(\delta+1)$ times. However, this code is inefficient because it requires $\delta n$ redundant bits, i.e., a redundancy that is linear in $n$. Helberg codes [22, 23] are a generalization of VT codes for multiple deletions. These codes can correct multiple deletions but their redundancy is at least linear in $n$ even for two deletions. Some of the works in the literature also studied the problem in the regime where the number of deletions is a constant fraction of $n$ [24-27] (rather than a constant number). The codes constructed for this regime have $\mathcal{O}(n)$ redundancy. In a recent work [28], Brakensiek et al. constructed zero-error codes with logarithmic redundancy, and polynomial time encoding and decoding, that can decode $\delta$ deletions ( $\delta$ fixed) in so-called "pattern rich" strings, which are a special family of strings that
exist for large block lengths (beyond $10^{6}$ bits for 2 deletions).
Following our work in this dissertation on unrestricted deletions, there has been some recent works in the literature which construct near-optimal codes for correcting deletions with zeroerror [29-31]. There has also been multiple subsequent works which study codes that can correct multiple deletions with low probability of error [27,32-38].

A series of works also study the problem of file synchronization for different edit models [1$3,5,6]$. In our work we focus on the following model which was studied by Venkataramanan et al. in $[1,5]$. In $[1,5]$, the authors assume that the file is affected by $t=o(n / \log n)$ deletions and insertions that are uniformly distributed within the file of size $n$. The authors introduce an interactive synchronization algorithm for synchronizing a file from deletions and insertions that is based on a divide-and-conquer approach. The key idea of this algorithm is to use center bits to divide a large file, affected by $d$ deletions, into shorter segments, such that each segment is affected by at most one deletion. Then, the authors exploit the properties of VT codes to correct single deletions in these shorter segments.

### 1.2.2 Contributions of Chapter 2

Most of the works mentioned in the previous section focus on constructing zero-error codes, which corresponds to correcting all possible deletions patterns for any codeword. In this dissertation, motivated by probabilistic file synchronization algorithms, we relax the zero-error decoding requirement and construct codes that can decode successfully for most deletion patterns and codewords. Furthermore, the existing codes for multiple deletions are non-systematic and non-linear ${ }^{1}$. As previously mentioned, in this dissertation, we are interested in systematic codes with low redundancy since they can be applied to file synchronization.

We design new systematic codes, that we call Guess \& Check (GC) codes, that can correct multiple deletions, with high probability. The high-level idea of GC codes is to treat bit deletions as symbol erasures over a higher field. First, we divide the string into adjacent binary blocks

[^1]which are mapped to symbols over a higher field $G F(q)$. Then, we encode the resulting $q$-ary string using a systematic MDS code which adds multiple parity symbols. To decode the string we use the following guess-and-check approach.

1. The guessing part: we go over all possibilities of the deletion locations among the blocks. We treat the affected blocks as hypothetical erasures and decode them using the MDS code. This results in multiple guesses out of which only one is the actual codeword.
2. The checking part: we use additional MDS parities (not used for erasure decoding) to check if the guesses are consistent with these parities. Decoding is successful if only one guess is consistent with all the parities.

We explain this more in Example 1.2. This example is for $\delta=1$ deletion for the sake of simplicity. GC codes can be generalized to $\delta>1$ deletions.

Example 1.1 (Encoding and Decoding Example of GC codes). Consider the binary message $\mathbf{u}$ of length $k=16$ given by

$$
\mathbf{u}=111100000011010001 .
$$

To encode this message, we first divide it into blocks of size $\log k=4$ bits, and map these blocks into symbols in $G F(17)^{2}$.

Then, we encode the resulting string in $G F(17)$ using a systematic $(6,4) M D S$ code. Suppose the encoding vectors use are $(1,1,1,1)$ and $(1,2,4,8)$. Hence, we obtain

$$
(14,0,13,1,11,6) \in G F(17)
$$

[^2]The corresponding binary codeword $x$ of length $n=k+2 \log k=24$ bits is given by

$$
x=\underset{111000001101000110110110}{\text { systematic bits }} \text { parity bits }
$$

Now, assume for the sake of simplicity that the deletion affects only the systematic bits and the decoder always recovers the parity bits successfully (we drop this assumption in general). Suppose that the $14^{\text {th }}$ bit of $x$ gets deleted. To decode the information, the decoder takes 4 guesses. In the first guess, it assumes that the deletion has affected the first block. The decoder sets the block boundaries according to this assumption, and considers that block 1 has been erased (the erasure is denoted by $\mathcal{E}$ ).

$$
\underbrace{111}_{\mathcal{E}} \underbrace{0000}_{0} \underbrace{0110}_{6} \underbrace{1001}_{10} .
$$

Then, the decoder uses the first parity symbol 11 to decode the erasure over $G F(17)$. The same steps are applied for guesses 2,3 , and 4, where in guess 2 the decoder assumes that the deletion has affected block 2, etc. The guessing phase results in 4 candidate strings: $Y_{1}=(12,0,6,10)$, $Y_{2}=(14,15,6,10), Y_{3}=(14,0,4,10)$, and $Y_{4}=(14,0,13,1)$.

In the checking phase, the decoder checks if the guesses are consistent with the second parity symbol 6. For instance, $Y_{1}$ is not consistent with the second parity since

$$
(12,0,6,10)(1,2,4,8)^{T}=14 \neq 6
$$

so $Y_{1}$ can be eliminated from the list of candidates strings. Similarly, the second guess $Y_{2}$ and third guess $Y_{3}$ can also be eliminated. The fourth guess $Y_{4}$ is the only guess that is consistent with the second parity, so the decoder declares successful decoding. Note that $x$ decoded as the binary equivalent of $Y_{4}$.

In Example 1.2, the decoding was successful for the case where the information message is $\mathbf{u}=1110000011010001$ and the deletion position is 14 . In general, we can have cases where
more than one guess is consistent with the parities. For instance, by running a simulation on a personal computer, we can find that in the case where the message is $\mathbf{u}=1110000011010001$ and the deletion position is 7 , the second and the fourth guesses, $Y_{2}=(14,14,9,10)$ and $Y_{4}=(14,13,2,8)$ are both consistent with the second parity 10 . Here, the decoder cannot tell which guess corresponds to the actual codeword, so it declares a decoding failure. Our simulations show that the event where the decoding fails is rare. In fact, we simulated all possible 16 -bit messages and all possible single deletion positions, and the results show that the GC code in Example 1.2 can decode successfully in roughly $98 \%$ of the cases. Our main theoretical contribution is Theorem 2.1 in Chapter 2, which we state informally below. The importance of this theorem is that it shows that the probability of a decoding failure vanishes asymptotically in $k$.

Example 1.2 (Encoding and Decoding Example of GC codes). Consider the binary message $\mathbf{u}$ of length $k=16$ given by

$$
\mathbf{u}=1000100000100010110
$$

To encode this message, we first divide it into blocks of size $\log k=4$ bits, and map these blocks into symbols in $G F\left(2^{4}\right)$ with primitive element $\alpha$, such that $\alpha^{4}=\alpha+1$.

$$
\mathbf{u}=\underbrace{\underbrace{\text { block 1 }} 1}_{\alpha^{14}} \underbrace{0^{\text {block } 2}}_{0} \underbrace{0000}_{\alpha^{14}} \underbrace{1001}_{\alpha^{5}} .
$$

Then, we encode the resulting string in $G F\left(2^{4}\right)$ using a systematic $(6,4) M D S$ code. Suppose the MDS encoding vectors use are $(1,1,1,1)$ and $\left(1, \alpha, \alpha^{2}, \alpha^{3}\right)$. Hence, we obtain

$$
\left(\alpha^{14}, 0, \alpha^{14}, \alpha^{5}, \alpha^{5}, \alpha^{11}\right)
$$

The corresponding binary codeword $x$ of length $n=k+2 \log k=24$ bits is given by

$$
x=\underset{100100001001011001101110}{\text { systematic bits }} .
$$

Now, assume for the sake of simplicity that the deletion affects only the systematic bits and that the decoder always recovers the parity bits successfully (we drop this assumption in general). Suppose that the $14^{\text {th }}$ bit of $x$ gets deleted. To decode the information, the decoder takes four guesses. In the first guess, it assumes that the deletion has affected the first block. The decoder sets the block boundaries according to this assumption, and considers that block 1 has been erased (the erasure is denoted by $\mathcal{E}$ ).

$$
\underbrace{100}_{\mathcal{E}} \underbrace{1000}_{\alpha^{3}} \underbrace{0100}_{\alpha^{2}} \underbrace{1010}_{\alpha^{9}} .
$$

Then, the decoder uses the first parity symbol $\alpha^{5}$ to decode the erasure over $G F\left(2^{4}\right)$. The same steps are applied for guesses 2,3 , and 4, where in guess 2 the decoder assumes that the deletion has affected block 2, etc. The guessing phase results in 4 candidate strings: $Y_{1}=\left(0, \alpha^{3}, \alpha^{2}, \alpha^{9}\right)$, $Y_{2}=\left(\alpha^{14}, 1, \alpha^{2}, \alpha^{9}\right), Y_{3}=\left(\alpha^{14}, 0, \alpha^{8}, \alpha^{9}\right)$, and $Y_{4}=\left(\alpha^{14}, 0, \alpha^{14}, \alpha^{5}\right)$.

In the checking phase, the decoder checks if the guesses are consistent with the second parity symbol $\alpha^{11}$. For instance, $Y_{1}$ is not consistent with the second parity since

$$
\left(0, \alpha^{3}, \alpha^{2}, \alpha^{9}\right)\left(1, \alpha, \alpha^{2}, \alpha^{3}\right)^{T}=\alpha^{12} \neq \alpha^{11}
$$

so $Y_{1}$ can be eliminated from the list of candidates strings. Similarly, the second guess $Y_{2}$ and third guess $Y_{3}$ can also be eliminated. The fourth guess $Y_{4}$ is the only guess that is consistent with the second parity, so the decoder declares successful decoding. Note that $x$ decoded as the binary equivalent of $Y_{4}$.

In Example 1.2, the decoding was successful for the case where the information message is $\mathbf{u}=1001000010010110$ and the deletion position is 14 . In general, we can have cases where
more than one guess is consistent with the parities. For instance, by running a simulation on a personal computer for the same code used in Example 1.2, we can find that in the case where the message is $\mathbf{u}=1101000010000101$ and the deletion position is 14 , the first and the fourth guesses, $Y_{1}=\left(\alpha^{13}, \alpha^{3}, \alpha^{2}, 1\right)$ and $Y_{4}=\left(\alpha^{13}, 0, \alpha^{3}, \alpha^{8}\right)$ are both consistent with the second parity $\alpha^{8}$. Here, the decoder cannot tell which guess corresponds to the actual codeword, so it declares a decoding failure. Our simulations show that the event where the decoding fails is rare. In fact, we simulated all possible 16-bit messages and all possible single deletion positions, and the results show that the GC code in Example 1.2 can decode successfully in roughly $98 \%$ of the cases. Our main theoretical contribution is Theorem 2.1 in Chapter 2, which we state informally below. The importance of this theorem is that it shows that the probability of a decoding failure vanishes asymptotically in $k$.

Theorem 2.1 (Informal). Guess $\mathcal{E B C}^{\text {Check }(G C) ~ c o d e s ~ a r e ~ s y s t e m a t i c ~ d e l e t i o n ~ c o r r e c t i n g ~ c o d e s ~}$ that have the following properties:

1. Redundancy: $c(\delta+1) \log k$ bits.,
2. Encoding and decoding complexity are polynomial in $k$,
3. For $c>2 \delta$, the probability of decoding failure vanishes asymptotically in $k$ for a uniform i.i.d. message and any $\delta$ deletion positions chosen independently of the codeword,
where $k$ is the length of information message, $\delta$ is the number of deletions, and $c$ is a code parameter representing the number of parity symbols.

The significance of the theorem is that GC codes can correct a constant number of deletions $\delta$, with high probability, and in polynomial time. Moreover, these codes are systematic and their redundancy is logarithmic in $k$. In Chapter 2, we provide a detailed theoretical analysis on these codes in addition to simulation results. We also show the following.

- The construction of GC codes is flexible in the sense that it enables interesting trade-offs between: redundancy, decoding complexity, and probability of decoding failure.
- GC codes can also correct $\delta$ insertions.
- We analyze the list decoding performance of GC codes and show that these codes can generate a small list of candidate strings in polynomial time.


### 1.2.3 Application of Guess \& Check Codes to File Synchronization

In this section, we discuss how we exploit the properties of GC codes to apply them to file synchronization. The existing multiple deletion correcting codes in the literature (e.g., [28,39, 40]) are non-systematic, and hence are inefficient for remote file synchronization, since using these codes would require sending the whole file in addition to the redundant bits. The systematic property of GC codes enables their application to file synchronization. We apply GC codes to the state of the art synchronization algorithm [1]. In [1], the two nodes communicate interactively to divide the large file, affected by $d$ deletions, into smaller segments, such that each segment is affected by at most one deletion. Then, VT codes are used to correct the single deletions. Now, consider a similar algorithm where the large file is divided such that the shorter segments are affected by $\delta(1<\delta \ll d)$ or fewer deletions. Then, use GC codes to correct the segments affected by more than one deletion ${ }^{3}$. Intuitively, isolating single deletions requires more communication rounds and a higher communication cost compared to isolating $\delta>1$ deletions. As a result, the application of GC codes results in savings in terms of number of communication rounds and also in terms of communication cost.

In Figure 1.1, we provide numerical simulations highlighting the savings obtained by using GC codes with $\delta=2$. These results are reproduced in the form a table in Chapter 2, Section 2.10. We refer to the algorithm in [1] by Sync-VT, and to the modified version by Sync-GC. The simulations are for a file of size 1 Mb with $d=300$ uniformly distributed deletions. The results show up to $73 \%$ improvement in number of communication rounds, and up to $14 \%$ improvement in communication cost.

[^3]

Figure 1.1: The results are averaged over 1000 runs. In each run, a string of size 1 Mb is chosen uniformly at random, and the file to be synchronized is obtained by deleting $d$ bits from it uniformly at random. The total communication cost is expressed in Kb .

### 1.3 Codes for Localized Deletions

In this section, we summarize our results in Chapter 3 on constructing codes for correcting localized deletions. In many applications, the deletion and insertion errors tend to occur in bursts (consecutive errors), or are localized within certain parts of the information. This happens for example when synchronization is lost for a certain period of time, resulting in a series of errors within that time frame. Another example is in file synchronization, in applications such as cloud computing, where large files are often edited by deleting and inserting characters in a relatively small part of the text (such as editing a paragraph). Motivated by these applications, we study the problem of constructing efficient codes that can correct localized deletions. The model that we consider is when $\delta \leq w$ deletions are localized in a window of size $w$ in the string, where the location of the window is unknown a priori. This model is a generalization of the bursty model in which all deletions must occur in consecutive positions.

### 1.3.1 Related Work on Bursts of Deletions

There are several works in the literature that study the problem of constructing codes that can correct a burst of $\delta$ deletions in a codeword of length $n$. Levenshtein [41] showed that the optimal number of redundant bits needed to correct a burst of size $\delta$ is $\log n+\delta-1$ asymptotically. Levenshtein [41] also constructed asymptotically optimal codes that can correct
a burst of at most two deletions. Cheng et al. [42] provided three constructions of codes that can correct a burst of exactly $\delta>2$ deletions. The lowest redundancy achieved by the codes in [42] is $\delta(\log (n / \delta+1))$. The fact that the number of deletions in the burst is exactly $\delta$, as opposed to at most $\delta$, is a crucial factor in the code constructions in [42]. Schoeny et al. [43] proved the existence of codes that can correct a burst of exactly $\delta$ deletions and have at most $\delta \log (n)+(\delta-1) \log (\log n)+\delta-1$ redundancy, for sufficiently large $n$. Schoeny et al. [43] also constructed codes that can correct a burst of at most $\delta$ deletions. Their results for the latter case improve on a previous result by Bours in [44].

Localized deletions are a generalization of bursty deletions. In the localized deletions model, the $\delta$ deletions are not necessarily consecutive but are localized in a window of size at most $w$ bits. Localized deletions have only been studied by Schoeny et al. in [43] for the the particular case of a single window of size $w=3$ or $w=4$.

### 1.3.2 Contributions of Chapter 3

In our work, we design novel codes for localized deletions that can be applied for arbitrary $w$, where $w=o(k)$ grows with the length of the information message $k$. Note that in the aformentioned works for the bursty model, the number of deletions is considered to be constant (fixed), i.e., $\delta$ does not grow with $n$. Our main contributions are Theorems 3.1 and 3.2 in Chapter 3 . We state Theorem 3.2 informally below.

Theorem 3.2 (Informal). For $\delta \leq w$ localized deletions, with $w=\Omega(\log k), G C$ codes have the following properties:

1. Redundancy: $(c+1) w+1$ bits,
2. Encoding complexity is subquadratic, and decoding complexity is quadratic,
3. For $c \geq 4$, the probability of decoding failure vanishes asymptotically in $k$,
where $c$ is a code parameter that represents the number of parity symbols.

The theorem shows that the redundancy of these codes is $(c+1) w+1$ for $w=\Omega(\log k)$, which is lower than the codes existing in the literature for bursts of fixed length [42-44]. These codes have an asymptotically vanishing probability of failure. Furthermore, the encoding and decoding algorithms of these codes are efficient since the encoding complexity is subquadratic and the decoding complexity is quadratic. We implement these codes and provide simulation results on their performance. We also extend our study to deletions that are localized within multiple windows in the codeword.

### 1.4 Codes for Trace Reconstruction

In this section, we summarize our results in Chapter 4 on coded trace reconstruction which has applications to DNA-based storage. DNA-based storage systems are celebrated for their ultrahigh storage densities that are of the order of $10^{15}-10^{20}$ bytes per gram of DNA [45]. These systems also introduce various challenges including but not limited to data degradation due to DNA aging and errors introduced during DNA sequencing. The aforementioned challenges often lead to deletion, insertion, and substitution errors in the stored data $[7,8]$. For instance, DNA sequencing with nanopores results in multiple erroneous reads of the data. As a result, recovering the data in question can be cast in the setting of trace reconstruction which is a well-studied problem in the computer science community. In trace reconstruction, the goal is to reconstruct an unknown sequence $x$ given random traces of $x$, where each trace is generated independently by randomly deleting and inserting symbols in $x$. Ultimately, one would like to devise strategies that allow the reconstruction of $x$ with minimum number of traces.

DNA synthesis strategies enable encoding arbitrary digital information and storing them in DNA [46-48]. Motivated by the application to DNA-based storage, the problem of coded trace reconstruction was introduced in [8]. In the coded version of the problem, one gets to encode the data before observing multiple traces of it, in an attempt to enhance the reconstruction process. Encoding data before storing it is a natural and widely-used strategy to ensure data reliability. However, employing this strategy in DNA-based storage systems requires novel techniques due
to the uniqueness of such systems in terms of the methods used to read the data and also the types of errors that are experienced. The goal is to design efficient codes that have a low redundancy which enable reconstructing a sequence from a small number of traces.

### 1.4.1 Related Work on Trace Reconstruction

Trace reconstruction was initially introduced in [49], motivated by problems in sequence alignment. In the trace reconstruction setting, we have multiple erroneous copies, called traces, of a sequence $x$ of length $n$. The traces are obtained by sending $x$ through independent random deletion channels, where each channel deletes each bit of $x$ independently and with probability $p$. The goal is to devise algorithms that allow the reconstruction of $x$ from few traces. There has been lots of works in the literature which study trace reconstruction in the case where $x$ is uncoded. These works are divided into two main perspectives. A series of works study this problem in the worst case setting, where the goal is to design an algorithm that can reconstruct any arbitrary sequence, with high probability [49-53]. Another line of work considers the average case, where the reconstruction algorithm is required to work with high probability for a random sequence $x[49-51,54-56]$.

In [49], the authors showed that for worst-case trace reconstruction with $p=\Theta(1 / \sqrt{n})$, the number of traces required to reconstruct $x$ with high probability is $\mathcal{O}(n \log n)$. The authors in [49] also showed that for average-case reconstruction with $p=1 / \log n$, the number traces required is $\mathcal{O}(\log n)$. As for the regime where $p$ is fixed, i.e., $p=\varepsilon$ with $0<\varepsilon<1$, for worst-case trace reconstruction, the state of the art reconstruction algorithm requires $\exp \left(\mathcal{O}\left(n^{1 / 3}\right)\right)$ traces to reconstruct $x$ with high probability $[52,53]$. Whereas, the state of the art lower bound for worst-case trace reconstruction, shows that the required number of traces is at least $n^{3 / 2}$ [57]. As for average-case trace reconstruction with $p=\varepsilon$, the best upper bound on the number of traces is $\exp \left(\log ^{1 / 3} n\right)$ [56], while the state of the art lower bound is $\log ^{5 / 2} n$ traces [57]. These results show that there is a significant gap between the upper and lower bounds for this problem for $p=\varepsilon$.

In this dissertation, we study the trace reconstruction problem in the coded setting, where we are allowed to code the sequence $x$ before observing multiple traces of it. The study of coded trace reconstructed was initiated by Cheraghci et al. in [8]. This study is motivated by applications to DNA-based storage systems. In such systems, DNA-sequencing using nanopores results in multiple erroneous copies of the data. In this setting, coding the stored data $x$ and devising efficient coded trace reconstruction algorithms can be used as a tool to enhance the reliability of DNA-based storage systems. It is also possible to use a deletion correcting code which would only require a single trace. However, using a deletion correcting code to correct $\delta$ deletions in the stored data would optimally require a redundancy that is $\Theta(\delta \log (n / \delta))$. Therefore, our goal in coded trace reconstruction, is to exploit the presence of multiple traces to reduce the redundancy in the stored data to $o(\delta \log (n / \delta))$.

The work in [8] studies coded trace reconstruction in the regime where $p=\varepsilon$ is fixed, i.e., a linear number of bits are deleted on average. The results in [8] show that by using a redundancy that is strictly smaller than the optimal redundancy of deletion correcting codes, one can efficiently reconstruct a coded sequence $x$ from $\log ^{c} n$ traces, with $c>1$. The work by Abroshan et al. in [58] studies the regime where the number of deletions $\delta$ is fixed with respect to $n$, and the positions of the deletions are uniformly random. The high-level idea of the approach in [58] is to concatenate a series of Varshamov-Tenengolts (VT) [18] codewords. The number of concatenated codewords is chosen to be a constant that is strictly greater than $\delta$, so that the average number of deletions per VT codeword is strictly less than 1. The reconstruction algorithm then decodes each VT codeword by either finding a deletion-free copy of it among the traces, or by decoding a single deletion using the VT decoder. Since the length of each codeword is a constant fraction of $n$, and since the redundancy of VT codes is logarithmic in the blocklength, then the resulting redundancy of the codes in [58] is $\Theta(\log n)$.

### 1.4.2 Contributions of Chapter 4

In our work, we study the regime where the bit deletion probability satisfies $p=\Theta(1 / n)$, which is similar to the regime in [58] since the average number of deletions for $p=\Theta(1 / n)$ is constant. We introduce novel codes for trace reconstruction. Our main contribution is Theorem 4.5 in Chapter 4. The significance of the theorem is that our codes can reconstruct a binary sequence from a constant number of traces, with constant redundancy. Hence, our codes have $\mathcal{O}(1)$ redundancy, which is significantly lower than the $\Theta(\log n)$ redundancy of the codes in [58]. Furthermore, the computational complexity of our reconstruction algorithm is linear $\mathcal{O}(n)$. We implement our codes and provide numerical simulations that validate our theoretical results.

### 1.5 Dissertation Outline

The dissertation is organized as follows. In Chapter 2, we present our construction of GC codes for unrestricted deletions and insertions, and describe their application to file synchronization. In Chapter 3, we discuss our code construction for the case of localized deletions. In Chapter 4, we present novel codes for trace reconstruction. We conclude with some open problems and proposed future work in Chapter 5.

## Chapter 2

## Guess \& Check Codes for

## Unrestricted Deletions

### 2.1 Introduction

In this chapter, we construct systematic codes that can correct multiple deletions with logarithmic redundancy. The systematic property of these codes enables their application to remote file synchronization. Since file synchronization algorithms are probabilistic in general (e.g., $[1-3,5,6]$ ), we relax the zero-error (worst-case) requirement and construct codes that have a low probability of decoding failure ${ }^{1}$. To this end, we assume that the information message is uniform iid, and that the positions of the deletions are independent of the codeword, i.e., oblivious deletions ${ }^{2}$.

Namely, our contributions in this chapter are the following:

[^4]1. We construct new explicit systematic codes, which we call Guess \& Check (GC) codes, that can correct, with high probability, and in polynomial time, up to a constant number of deletions $\delta$ (or insertions) occurring in a binary string. The encoding and decoding algorithms of GC codes are deterministic. Moreover, these codes have logarithmic redundancy of value $n-k=c(\delta+1) \log k$, where $k$ and $n$ are the lengths of the message and the codeword, respectively, and $c>\delta$ is a code parameter that is independent of $k$.
2. GC codes enable different trade-offs between redundancy, decoding complexity, and probability of decoding failure.
3. We implemented GC codes and the programming code can be found and tested online on the link in [62]. Based on our implementations, we provide numerical simulations on the decoding failure of GC codes and compare these simulations to our theoretical results. For instance, we observe that a GC code with rate 0.8 can correct up to 4 deletions in a message of length 1024 bits with no decoding failure detected within 10000 runs of simulations.
4. We demonstrate how GC codes can be used as a building block for file synchronization algorithms by including these codes as part of the synchronization algorithm proposed by Venkataramanan et al. in [1,5]. As a result, we provide numerical simulations highlighting the savings in number of rounds and total communication cost.
5. We study the list decoding properties of GC codes and show that the GC decoder can generate a small list of candidate strings in polynomial time.

The chapter is organized as follows. In Section 2.2, we introduce the necessary notations used throughout the chapter. We state and discuss the main results of this chapter in Section 2.3. In Section 2.4, we provide encoding and decoding examples on GC codes. In Section 2.5, we describe in detail the encoding and decoding schemes of GC codes. The proof of the main result of this chapter is given in Section 2.6. In Section 2.7, we explain the trade-offs achieved by GC codes. In Section 2.8, we explain how these codes can be used to correct $\delta$ insertions instead


Figure 2.1: General encoding block diagram of GC codes for $\delta$ deletions. Block I: The binary message of length $k$ bits is chunked into adjacent blocks of length $\log k$ bits each, and each block is mapped to its corresponding symbol in $G F(q)$ where $q=2^{\log k}=k$. Block II: The resulting string is coded using a systematic $(k / \log k+c, k / \log k) q$-ary MDS code where $c>\delta$ is the number of parity symbols. Block III: The symbols in $G F(q)$ are mapped to their binary representations. Block IV: Only the parity bits are coded using a $(\delta+1)$ repetition code.
of $\delta$ deletions. The results of the numerical simulations on the decoding failure of GC codes and their application to file synchronization are shown in Sections 2.9 and 2.10, respectively. The list decoding properties of GC codes are discussed in 2.11. We conclude with some open problems in Section 2.12.

### 2.2 Notation

Let $k$ and $n$ be the lengths in bits of the message and codeword, respectively. Let $\delta$ be a constant representing the number of deletions. Without loss of generality, we assume that $k$ is a power of 2 . GC codes are based on $q$-ary systematic $(\lceil k / \log k\rceil+c,\lceil k / \log k\rceil)$ MDS codes, where $q=k>\lceil k / \log k\rceil+c$ and $c>\delta$ is a code parameter representing the number of MDS parity symbols ${ }^{3}$. We will drop the ceiling notation for $\lceil k / \log k\rceil$ and simply write $k / \log k$. All logarithms in this chapter are of base 2. The block diagram of the encoder is shown in Fig. 2.1. We denote binary and $q$-ary vectors by lower and upper case bold letters respectively, and random variables by calligraphic letters.

[^5]
### 2.3 Main Result

Let $\mathbf{u}$ be a binary vector of length $k$ with iid $\operatorname{Bernoulli}(1 / 2)$ components representing the information message. The message $\mathbf{u}$ is encoded into the codeword $\mathbf{x}$ of length $n$ bits using the Guess \& Check (GC) code illustrated in Fig. 2.1. The GC decoder, explained in Section 2.5, can either decode successfully and output the decoded string, or output a "failure to decode" error message because it cannot make a correct decision. The latter case is referred to as a decoding failure, and its corresponding event is denoted by $F$.

Theorem 2.1. Guess $\xi^{3}$ Check (GC) codes can correct in polynomial time up to a constant number of $\delta$ deletions occurring within $\mathbf{x}$. Let $c>\delta$ be an integer. The code has the following properties:

1. Redundancy: $n-k=c(\delta+1) \log k$ bits.
2. Encoding complexity is $\mathcal{O}(k \log k)$, and decoding complexity is $\mathcal{O}\left(\frac{k^{\delta+1}}{\log ^{\delta-1} k}\right)$.
3. For any $\delta$ deletion positions chosen independently of $\mathbf{x}$, the probability that the decoding fails for a uniform iid message is: $\operatorname{Pr}(F)=\mathcal{O}\left(\frac{k^{2 \delta-c}}{\log ^{\delta} k}\right)$.

The result on the probability of decoding failure in Theorem 2.1 applies for any $\delta$ deletion positions which are chosen independently of the codeword x. Hence, the same result can be also obtained for any random distribution over the $\delta$ deletion positions (like the uniform distribution for example), by averaging over all the possible positions of the $\delta$ deletions.

GC codes enable trade-offs between the code properties shown in Theorem 2.1, this will be highlighted later in Section 2.7. The code properties show the following: (i) the redundancy is logarithmic in $k$, and the code rate, $R=k /(k+c(\delta+1) \log k)$, is asymptotically optimal in $k$ (approaches 1 as $k$ goes to infinity); (ii) the order of complexity is polynomial in $k$ and is not affected by the integer $c$; (iii) the probability of decoding failure goes to zero polynomially in $k$ if $c>2 \delta$; and exponentially in $c$ for a fixed $k$. Note that the decoder can always detect when it cannot decode successfully. This can serve as an advantage in models which allow feedback. There, the decoder can ask for additional redundancy to be able to decode successfully.

### 2.4 Encoding and Decoding Examples

Guess \& Check (GC) codes can correct up to $\delta$ deletions with high probability. We provide examples to illustrate the encoding and decoding schemes. The examples are for $\delta=1$ deletion just for the sake of simplicity ${ }^{4}$.

Example 2.1 (Encoding). Consider a binary message u of length $k=16$ given by

$$
\mathbf{u}=111100000011010001 .
$$

$\mathbf{u}$ is encoded by following the different encoding blocks illustrated in Fig. 2.1.

1) Binary to $q$-ary (Block I, Fig. 2.1). The message $\mathbf{u}$ is chunked into adjacent blocks of length $\log k=4$ bits each,

$$
\mathbf{u}=\underbrace{\stackrel{\text { block 1 }}{ }_{1110}^{\underbrace{\text { block 2 }}_{0}} \underbrace{0000}_{\alpha^{13}} \underbrace{110 \text { block 3 }}_{1}}_{\alpha^{11}} \underbrace{0001}_{\underbrace{\text { block 4 }}_{1}} .
$$

Each block is then mapped to its corresponding symbol in $G F(q), q=k=2^{4}=16$, by considering its leftmost bit as its most significant bit. This results in a string $\mathbf{U}$ which consists of $k / \log k=4$ symbols in $G F(16)$. The extension field used here has a primitive element $\alpha$, with $\alpha^{4}=\alpha+1$. Hence, $\mathbf{U} \in G F(16)^{4}$ is given by

$$
\mathbf{U}=\left(\alpha^{11}, 0, \alpha^{13}, 1\right)
$$

2) Systematic MDS code (Block II, Fig. 2.1). U is then coded using a systematic ( $k / \log k+$ $c, k / \log k)=(6,4)$ MDS code over $G F(16)$, with $c=2>\delta$. The encoded string is denoted by $\mathbf{X} \in G F(16)^{6}$ and is given by multiplying $\mathbf{U}$ by the following code generator matrix,

$$
\mathbf{X}=\left(\alpha^{11}, 0, \alpha^{13}, 1\right)\left(\begin{array}{cccccc}
1 & 0 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & \alpha \\
0 & 0 & 1 & 0 & 1 & \alpha^{2} \\
0 & 0 & 0 & 1 & 1 & \alpha^{3}
\end{array}\right)=\left(\alpha^{11}, 0, \alpha^{13}, 1, \alpha, \alpha^{10}\right)
$$

3) $Q$-ary to binary (Block III, Fig. 2.1). The binary codeword corresponding to $\mathbf{X}$, of length

[^6]$n=k+2 \log k=24$ bits, is
$$
\mathbf{x}=\underbrace{1110} \underbrace{0000} \underbrace{1101} \underbrace{0001} \underbrace{0010} \underbrace{0111} .
$$

For simplicity we skip the last encoding step (Block IV) intended to protect the parity bits and assume that deletions affect only the systematic bits.

The high level idea of the decoding algorithm is to: (i) make an assumption on in which block the bit deletion has occurred (the guessing part); (ii) chunk the bits accordingly, treat the affected block as erased, decode the erasure and check whether the obtained sequence is consistent with the parities (the checking part); (iii) go over all the possibilities.

Example 2.2 (Successful Decoding). Suppose that the $14^{\text {th }}$ bit of $\mathbf{x}$ gets deleted,

$$
\mathbf{x}=1110000011010 \underline{0} 0100100111 .
$$

The decoder receives the following 23 bit string $\mathbf{y}$,

$$
\mathbf{y}=11100000110100100100111 .
$$

The decoder goes through all the possible $k / \log k=4$ cases, where in each case $i, i=1, \ldots, 4$, the deletion is assumed to have occurred in block $i$ and $\mathbf{y}$ is chunked accordingly. Given this assumption, symbol $i$ is considered erased and erasure decoding is applied over $G F(16)$ to recover this symbol. Furthermore, given two parities, each symbol i can be recovered in two different ways. Without loss of generality, we assume that the first parity $p_{1}, p_{1}=\alpha$, is the parity used for decoding the erasure. The decoded $q$-ary string in case $i$ is denoted by $\mathbf{Y}_{\mathbf{i}} \in G F(16)^{4}$, and its binary representation is denoted by $\mathbf{y}_{\mathbf{i}} \in G F(2)^{16}$. The four cases are shown below:

Case 1: The deletion is assumed to have occurred in block 1, so $\mathbf{y}$ is chunked as follows and the
erasure is denoted by $\mathcal{E}$,

$$
\underbrace{111}_{\mathcal{E}} \underbrace{0000}_{0} \underbrace{0110}_{\alpha^{5}} \underbrace{1001}_{\alpha^{14}} \underbrace{0010}_{\alpha} \underbrace{0111}_{\alpha^{10}} .
$$

Applying erasure decoding over $G F(16)$, the recovered value of symbol 1 is $\alpha^{13}$. Hence, the decoded $q-$ ary string $\mathbf{Y}_{\mathbf{1}} \in G F(16)^{4}$ is

$$
\mathbf{Y}_{\mathbf{1}}=\left(\alpha^{13}, 0, \alpha^{5}, \alpha^{14}\right)
$$

Its equivalent in binary $\mathbf{y}_{\mathbf{1}} \in G F(2)^{16}$ is

$$
\mathbf{y}_{\mathbf{1}}=\underbrace{1101}_{\alpha^{13}} \underbrace{0000}_{0} \underbrace{0110}_{\alpha^{5}} \underbrace{1001}_{\alpha^{14}} .
$$

Now, to check our assumption, we test whether $\mathbf{Y}_{\mathbf{1}}$ is consistent with the second parity $p_{2}=\alpha^{10}$. However, the computed parity is

$$
\left(\alpha^{13}, 0, \alpha^{5}, \alpha^{14}\right)\left(1, \alpha, \alpha^{2}, \alpha^{3}\right)^{T}=\alpha \neq \alpha^{10}
$$

This shows that $\mathbf{Y}_{\mathbf{1}}$ does not satisfy the second parity. Therefore, we deduce that our assumption on the deletion location is wrong, i.e., the deletion did not occur in block 1. Throughout the chapter we refer to such cases as impossible cases.

Case 2: The deletion is assumed to have occurred in block 2, so the sequence is chunked as follows

$$
\underbrace{1110}_{\alpha^{11}} \underbrace{000}_{\mathcal{E}} \underbrace{0110}_{\alpha^{5}} \underbrace{1001}_{\alpha^{14}} \underbrace{0010}_{\alpha} \underbrace{0111}_{\alpha^{10}} .
$$

Applying erasure decoding, the recovered value of symbol 2 is $\alpha^{4}$. Now, before checking whether the decoded string is consistent with the second parity $p_{2}$, one can notice that the binary representation of the decoded erasure (0011) is not a supersequence of the sub-block (000). So, without checking $p_{2}$, we can deduce that this case is impossible.

Definition 2.2. We restrict this definition to the case of $\delta=1$ deletion with two MDS parity symbols in $G F(q)$. A case $i, i=1,2, \ldots, k / \log k$, is said to be possible if it satisfies the two criteria below simultaneously.

Criterion 1: The $q$-ary string decoded based on the first parity in case $i$, denoted by $\mathbf{Y}_{\mathbf{i}}$, satisfies the second parity.

Criterion 2: The binary representation of the decoded erasure is a supersequence of its corresponding sub-block.

If any of the two criteria is not satisfied, the case is said to be impossible.

The two criteria mentioned above are both necessary. For instance, in this example, case 2 does not satisfy Criterion 2 but it is easy to verify that it satisfies Criterion 1. Furthermore, case 1 satisfies Criterion 1 but does not satisfy Criterion 2. A case is said to be possible if it satisfies both criteria simultaneously.

Case 3: The deletion is assumed to have occurred in block 3, so the sequence is chunked as follows

$$
\underbrace{1110}_{\alpha^{11}} \underbrace{0000}_{0} \underbrace{110}_{\mathcal{E}} \underbrace{1001}_{\alpha^{14}} \underbrace{0010}_{\alpha} \underbrace{0111}_{\alpha^{10}} .
$$

In this case, the decoded binary string is

$$
\mathbf{y}_{\mathbf{3}}=\underbrace{1110}_{\alpha^{11}} \underbrace{0000}_{0} \underbrace{0101}_{\alpha^{8}} \underbrace{1001}_{\alpha^{14}} .
$$

By following the same steps as cases 1 and 2, it is easy to verify that both criteria are not satisfied in this case, i.e., case 3 is also impossible.

Case 4: The deletion is assumed to have occurred in block 4, so the sequence is chunked as follows

$$
\underbrace{1110}_{\alpha^{11}} \underbrace{0000}_{0} \underbrace{1101}_{\alpha^{13}} \underbrace{001}_{\mathcal{E}} \underbrace{0010}_{\alpha} \underbrace{0111}_{\alpha^{10}} .
$$

In this case, the decoded binary string is

$$
\mathbf{y}_{4}=\underbrace{1110}_{\alpha^{11}} \underbrace{0000}_{0} \underbrace{1101}_{\alpha^{13}} \underbrace{0001}_{1} .
$$

Here, it is easy to verify that this case satisfies both criteria and is indeed possible.
After going through all the cases, case 4 stands alone as the only possible case. So the decoder declares successful decoding and outputs $\mathbf{y}_{\mathbf{4}}\left(\mathbf{y}_{\mathbf{4}}=\mathbf{u}\right)$.

The next example considers another message $\mathbf{u}$ and shows how the proposed decoding scheme can lead to a decoding failure. The importance of Theorem 2.1 is that it shows that the probability of a decoding failure vanishes a $k$ goes to infinity.

Example 2.3 (Decoding failure). Let $k=16, \delta=1$ and $c=2$. Consider the binary message $\mathbf{u}$ given by

$$
\mathbf{u}=1101000010000101 .
$$

Following the same encoding steps as before, the $q-a r y$ codeword $\mathbf{X} \in G F(16)^{6}$ is given by

$$
\mathbf{X}=\left(\alpha^{13}, 0, \alpha^{3}, \alpha^{8}, 0, \alpha^{8}\right)
$$

We still assume that the deletion affects only the systematic bits. Suppose that the $14^{\text {th }}$ bit of the binary codeword $\mathbf{x} \in G F(2)^{24}$ gets deleted

$$
\mathrm{x}=1101000010000 \underline{1} 0100000101 .
$$

The decoder receives the following 23 bit binary string $\mathbf{y} \in G F(2)^{23}$,

$$
\mathbf{y}=11101010000010000001000000101 .
$$

The decoding is carried out as explained in Example 3.3. The q-ary strings decoded in case 1
and case 4 are

$$
\begin{aligned}
& \mathbf{Y}_{\mathbf{1}}=\left(\alpha^{13}, \alpha^{3}, \alpha^{2}, 1\right), \\
& \mathbf{Y}_{\mathbf{4}}=\left(\alpha^{13}, 0, \alpha^{3}, \alpha^{8}\right) .
\end{aligned}
$$

It is easy to verify that both cases 1 and 4 are possible cases. The decoder here cannot know which of the two cases is the correct one, so it declares a decoding failure.

Remark 2.3. In the previous analysis, each case refers to the assumption that a certain block is affected by the deletion. Hence, among all the cases considered, there is only one correct case that corresponds to the actual deletion location. That correct case always satisfies the two criteria for possible cases (Definition 3.6). So whenever there is only one possible case (like in Example 3.3), the decoding will be successful since that case would be for sure the correct one. However, in general, the analysis may yield multiple possible cases. Nevertheless, the decoding can still be successful if all these possible cases lead to the same decoded string. An example of this is when the transmitted codeword is the all 0 's sequence. Regardless of the deletion position, this sequence will be decoded as all 0 's in all the cases. In fact, whenever the deletion occurs within a run of 0 's or 1's that extends to multiple blocks, the cases corresponding to these blocks will all be possible and lead to the same decoded string. However, sometimes the possible cases can lead to different decoded strings like in Example 2.3, thereby causing a decoding failure.

### 2.5 General Encoding and Decoding of GC Codes

In this section, we describe the general encoding and decoding schemes that can correct up to $\delta$ deletions. The encoding and decoding steps for $\delta>1$ deletions are a direct generalization of the steps for $\delta=1$ described in the previous section. For decoding, we assume without loss of generality that exactly $\delta$ deletions have occurred. Therefore, the length of the binary string $\mathbf{y}$ received by the decoder is $n-\delta$ bits.

### 2.5.1 Encoding Steps

The encoding steps follow from the block diagram that was shown in Fig. 2.1.

1) Binary to $q$-ary (Block I, Fig. 2.1). The message $\mathbf{u}$ is chunked into adjacent blocks of length $\log k$ bits each. Each block is then mapped to its corresponding symbol in $G F(q)$, where $q=2^{\log k}=k$. This results in a string $\mathbf{U}$ which consists of $k / \log k$ symbols in $G F(q)^{5}$.
2) Systematic MDS code (Block II, Fig. 2.1). U is then coded using a systematic ( $k / \log k+$ $c, k / \log k)$ MDS code over $G F(q)$, where $c>\delta$ is a code parameter. The $q$-ary codeword $\mathbf{X}$ consists of $k / \log k+c$ symbols in $G F(q)$.
3) $Q$-ary to binary (Block III, Fig. 2.1). The binary representations of the symbols in $\mathbf{X}$ are concatenated respectively.
4) Coding parity bits by repetition (Block IV, Fig. 2.1). Only the parity bits are coded using a $(\delta+1)$ repetition code, i.e., each bit is repeated $(\delta+1)$ times. The resulting binary codeword $\mathbf{x}$ to be transmitted is of length $n=k+c(\delta+1) \log k$.

### 2.5.2 Decoding Steps

1) Decoding the parity symbols of Block II (Fig. 2.1): these parities are protected by a $(\delta+1)$ repetition code, and therefore can be always recovered correctly by the decoder. A simple way to do this is to examine the bits of $\mathbf{y}$ from right to left and decode deletions instantaneously until the total length of the decoded sequence is $c(\delta+1)$ bits (the original length of the coded parity bits). Therefore, for the remaining steps we will assume without loss of generality that all the $\delta$ deletions have occurred in the systematic bits.
2) The guessing part: the number of possible ways to distribute the $\delta$ deletions among the $k / \log k$ blocks is

$$
t \triangleq\binom{k / \log k+\delta-1}{\delta}
$$

[^7]We index these possibilities by $i, i=1, \ldots, t$, and refer to each possibility by case $i$.
The decoder goes through all the $t$ cases (guesses).
3) The checking part: for each case $i, i=1, \ldots, t$, the decoder (i) chunks the sequence according to the corresponding assumption; (ii) considers the affected blocks erased and maps the remaining blocks to their corresponding symbols in $G F(q)$; (iii) decodes the erasures using the first $\delta$ parity symbols; (iv) checks whether the case is possible or not based on the criteria described below.

Definition 2.4. For $\delta$ deletions, a case $i, i=1,2, \ldots, t$, is said to be possible if it satisfies the two criteria below simultaneously.

Criterion 1: the decoded $q$-ary string in case $i$, denoted by $\mathbf{Y}_{\mathbf{i}} \in G F(q)^{k / \log k}$, satisfies the last $c-\delta$ parities simultaneously.

Criterion 2: The binary representations of all the decoded erasures in $\mathbf{Y}_{\mathbf{i}}$ are supersequences of their corresponding sub-blocks.

If any of these two criteria is not satisfied, the case is said to be impossible.
4) After going through all the cases, the decoder declares successful decoding if (i) only one possible case exists; or (ii) multiple possible cases exist but all lead to the same decoded string. Otherwise, the decoder declares a decoding failure.

### 2.6 Proof of Theorem 2.1

### 2.6.1 Redundancy

The $(k / \log k+c, k / \log k) q$-ary MDS code in step 2 of the encoding scheme adds a redundancy of $c \log k$ bits. These $c \log k$ bits are then coded using a $(\delta+1)$ repetition code. Therefore, the overall redundancy of the code is $c(\delta+1) \log k$ bits.

### 2.6.2 Complexity

i) Encoding Complexity: The complexity of mapping a binary string to its corresponding $q$-ary symbol (step 1), or vice versa (step 3 ), is $\mathcal{O}(k)$. Furthermore, the encoding complexity of a $(k / \log k+c, k / \log k) q$-ary systematic MDS code is quantified by the complexity of computing the $c$ MDS parity symbols. Computing one MDS parity symbol involves $k / \log k$ multiplications of symbols in $G F(q)$. The complexity of multiplying two symbols in $G F(q)$ is $\mathcal{O}\left(\log ^{2} q\right)$. Recall that in our code $q=k$. Therefore, the complexity of step 2 is $\mathcal{O}\left(\log ^{2} k \cdot c \cdot k / \log k\right)=\mathcal{O}(c \cdot k \log k)$. Step 4 in the encoding scheme codes $c \log k$ bits by repetition, its complexity is $\mathcal{O}(c \log k)$. Therefore, the encoding complexity of GC codes is $\mathcal{O}(c \cdot k \log k)=\mathcal{O}(k \log k)$ since $c>\delta$ is a constant.
ii) Decoding Complexity: The computationally dominant step in the decoding algorithm is step 3 , that goes over all the $t$ cases and decodes the erasures in each case ${ }^{6}$. Since the erasures to be decoded are within the systematic bits, then decoding $\delta$ erasures in each case can be done by performing the following steps: (a) multiplying the unerased systematic symbols by the corresponding $\delta$ MDS encoding vectors; (b) subtracting the obtained results from the values of the corresponding MDS parities; (c) inverting a $\delta \times \delta$ matrix. Since $\delta$ is a constant, the complexity of the previous steps is $\mathcal{O}\left(\log ^{2} q \cdot k / \log k\right)=\mathcal{O}(k \log k)$. Now since the erasure decoding is performed for all the $t$ cases, the total decoding complexity is $\mathcal{O}(t \cdot k \log k)$. The number of cases $t$ is given by

$$
\begin{equation*}
\binom{k / \log k+\delta-1}{\delta}=\mathcal{O}\left(\frac{k^{\delta}}{\log ^{\delta} k}\right) \tag{2.1}
\end{equation*}
$$

[^8](2.1) follows from the fact that $\binom{b}{a} \leq b^{a}$ for all integer values of $a$ and $b$ such that $1 \leq a \leq b$. Therefore, the overall decoding complexity is
$$
\mathcal{O}\left(\frac{k^{\delta+1}}{\log ^{\delta-1} k}\right)
$$
which is polynomial in $k$ for constant $\delta$.

### 2.6.3 Proof of the probability of decoding failure for $\delta=1$

To prove the upper bound on the probability of decoding failure, we first introduce the steps of the proof for $\delta=1$ deletion. Then, we generalize the proof to the case of $\delta>1$.

The probability of decoding failure for $\delta=1$ is computed over all possible $k$-bit messages. Recall that the bits of the message $\mathbf{u}$ are iid $\operatorname{Bernoulli}(1 / 2)$. The message $\mathbf{u}$ is encoded as shown in Fig. 2.1. For $\delta=1$, the decoder goes through a total of $k / \log k$ cases, where in a case $i$ it decodes by assuming that block $i$ is affected by the deletion. Let $\mathcal{Y}_{\boldsymbol{i}}$ be the random variable representing the $q$-ary string decoded in case $i, i=1,2, \ldots, k / \log k$, in step 3 of the decoding scheme. Let $\mathbf{Y} \in G F(q)^{k / \log k}$ be a realization of the random variable $\mathcal{Y}_{\boldsymbol{i}}$. We denote by $\mathcal{P}_{r} \in G F(q), r=1,2, \ldots, c$, the random variable representing the $r^{t h}$ MDS parity symbol (Block II, Fig. 2.1). Also, let $\mathbf{G}_{\mathbf{r}} \in G F(q)^{k / \log k}$ be the MDS encoding vector responsible for generating $\mathcal{P}_{r}$. Consider $c>\delta$ arbitrary MDS parities $p_{1}, \ldots, p_{c}$, for which we define the following sets. For $r=1, \ldots, c$,

$$
\begin{aligned}
& \mathrm{A}_{\mathrm{r}} \triangleq\left\{\mathbf{Y} \in G F(q)^{k / \log k} \mid \mathbf{G}_{\mathbf{r}}^{\mathbf{T}} \mathbf{Y}=p_{r}\right\} \\
& \mathrm{A} \triangleq \mathrm{~A}_{1} \cap \mathrm{~A}_{2} \cap \ldots \cap \mathrm{~A}_{\mathrm{c}}
\end{aligned}
$$

$\mathrm{A}_{\mathrm{r}}$ and A are affine subspaces of dimensions $k / \log k-1$ and $k / \log k-c$, respectively. Therefore,

$$
\begin{equation*}
\left|\mathrm{A}_{\mathrm{r}}\right|=q^{\frac{k}{\log k}-1} \text { and }|\mathrm{A}|=q^{\frac{k}{\log k}-c} \tag{2.2}
\end{equation*}
$$

Recall that the correct values of the MDS parities are recovered at the decoder, and that for $\delta=1, \mathcal{Y}_{i}$ is decoded based on the first parity. Hence, for a fixed MDS parity $p_{1}$, and for $\delta=1$ deletion, $\mathcal{Y}_{\boldsymbol{i}}$ takes values in $\mathrm{A}_{1}$. Note that $\boldsymbol{\mathcal { Y }}_{\boldsymbol{i}}$ is not necessarily uniformly distributed over $\mathrm{A}_{1}$. For instance, if the assumption in case $i$ is wrong, two different message inputs can generate the same decoded string $\mathcal{Y}_{i} \in \mathrm{~A}_{1}$. We illustrate this later through Example 2.4. The next claim gives an upper bound on the probability mass function of $\mathcal{Y}_{i}$ for $\delta=1$ deletion.

Claim 2.5. For any case $i, i=1,2, \ldots, k / \log k$,

$$
\operatorname{Pr}\left(\boldsymbol{\mathcal { Y }}_{i}=\mathbf{Y} \mid \mathcal{P}_{1}=p_{1}\right) \leq \frac{2}{q^{\frac{k}{\log _{k}}-1}} .
$$

Claim 2.5 can be interpreted as that at most 2 different input messages can generate the same decoded string $\mathcal{Y}_{\boldsymbol{i}} \in \mathrm{A}_{1}$. We assume Claim 2.5 is true for now and prove it later in Section 2.6.5. Next, we use this claim to prove the following upper bound on the probability of decoding failure for $\delta=1$,

$$
\begin{equation*}
\operatorname{Pr}(F)<\frac{2}{k^{c-2} \log k} . \tag{2.3}
\end{equation*}
$$

In the general decoding scheme, we mentioned two criteria which determine whether a case is possible or not (Definition 2.4). Here, we upper bound $\operatorname{Pr}(F)$ by taking into account Criterion 1 only. Based on Criterion 1, if a case $i$ is possible, then $\mathcal{Y}_{i}$ satisfies all the $c$ MDS parities simultaneously, i.e., $\mathcal{Y}_{i} \in A$. Without loss of generality, we assume case 1 is the correct case, i.e., the deletion occurred in block 1. A decoding failure is declared if there exists a possible case $j, j=2, \ldots, k / \log k$, that leads to a decoded string different than that of case 1. Namely,
$\mathcal{Y}_{j} \in \mathrm{~A}$ and $\mathcal{Y}_{j} \neq \mathcal{Y}_{1}$. Therefore,

$$
\begin{align*}
\operatorname{Pr}\left(F \mid \mathcal{P}_{1}=p_{1}\right) & \leq \operatorname{Pr}\left(\bigcup_{j=2}^{k / \log k}\left\{\mathcal{Y}_{j} \in \mathrm{~A}, \boldsymbol{\mathcal { Y }}_{\boldsymbol{j}} \neq \mathcal{Y}_{1}\right\} \mid \mathcal{P}_{1}=p_{1}\right)  \tag{2.4}\\
& \leq \sum_{j=2}^{k / \log k} \operatorname{Pr}\left(\boldsymbol{\mathcal { Y }}_{j} \in \mathrm{~A}, \boldsymbol{\mathcal { Y }}_{\boldsymbol{j}} \neq \mathcal{Y}_{\mathbf{1}} \mid \mathcal{P}_{1}=p_{1}\right)  \tag{2.5}\\
& \leq \sum_{j=2}^{k / \log k} \operatorname{Pr}\left(\boldsymbol{\mathcal { Y }}_{j} \in \mathrm{~A} \mid \mathcal{P}_{1}=p_{1}\right)  \tag{2.6}\\
& =\sum_{j=2}^{k / \log k} \sum_{\mathbf{Y} \in \mathrm{A}} \operatorname{Pr}\left(\boldsymbol{\mathcal { Y }}_{\boldsymbol{j}}=\mathbf{Y} \mid \mathcal{P}_{1}=p_{1}\right)  \tag{2.7}\\
& \leq \sum_{j=2}^{k / \log k} \sum_{\mathbf{Y} \in \mathrm{A}} \frac{2}{q^{\frac{k}{\log k}-1}}  \tag{2.8}\\
& =\sum_{j=2}^{k / \log k}|\mathrm{~A}| \frac{2}{q^{\frac{k}{\log }-1}}  \tag{2.9}\\
& =\sum_{j=2}^{k / \log k} 2 \frac{q^{\frac{k}{\log k}-c}}{q^{\frac{k}{\log k}-1}}  \tag{2.10}\\
& =\left(\frac{k}{\log k}-1\right) \frac{2}{q^{c-1}}  \tag{2.11}\\
& <\frac{2}{k^{c-2} \log k} . \tag{2.12}
\end{align*}
$$

(3.5) follows from applying the union bound. (3.6) follows from the fact that $\operatorname{Pr}\left(\boldsymbol{\mathcal { Y }}_{\boldsymbol{j}} \neq \boldsymbol{\mathcal { Y }}_{\mathbf{1}} \mid \mathcal{Y}_{j} \in \mathrm{~A}, \mathcal{P}_{1}=p_{1}\right) \leq$ 1. (3.8) follows from Claim 2.5. (3.10) follows from (3.3). (3.12) follows from the fact that $q=k$ in the coding scheme. Next, to complete the proof of (2.3), we use (3.12) and average over all values of $p_{1}$.

$$
\begin{aligned}
\operatorname{Pr}(F) & =\sum_{p_{1} \in G F(q)} \operatorname{Pr}\left(F \mid \mathcal{P}_{1}=p_{1}\right) \operatorname{Pr}\left(\mathcal{P}_{1}=p_{1}\right) \\
& <\sum_{p_{1} \in G F(q)} \frac{2}{k^{c-2} \log k} \operatorname{Pr}\left(\mathcal{P}_{1}=p_{1}\right) \\
& =\frac{2}{k^{c-2} \log k} .
\end{aligned}
$$

### 2.6.4 Proof of the probability of decoding failure for $\delta$ deletions

We now generalize the previous proof for $\delta>1$ deletions and show that

$$
\operatorname{Pr}(F)=\mathcal{O}\left(\frac{1}{k^{c-2 \delta} \log ^{\delta} k}\right)
$$

For $\delta$ deletions the number of cases is given by

$$
\begin{equation*}
t=\binom{k / \log k+\delta-1}{\delta}=\mathcal{O}\left(\frac{k^{\delta}}{\log ^{\delta} k}\right) \tag{2.13}
\end{equation*}
$$

Consider the random variable $\mathcal{Y}_{\boldsymbol{i}}$ which represents the $q$-ary string decoded in case $i, i=$ $1,2, \ldots, t$. The next claim generalizes Claim 2.5 for $\delta>1$ deletions.

Claim 2.6. There exists a deterministic function $h$ of $\delta, h(\delta)$ independent of $k$, such that for any case $i, i=1,2, \ldots, t$,

$$
\operatorname{Pr}\left(\mathcal{Y}_{i}=\mathbf{Y} \mid \mathcal{P}_{1}=p_{1}, \ldots, \mathcal{P}_{\delta}=p_{\delta}\right) \leq \frac{h(\delta)}{q^{\frac{k}{\log k}-\delta}}
$$

We assume Claim 2.6 is true for now and prove it later (see Appendix A). To bound the probability of decoding failure for $\delta$ deletions, we use the result of Claim 2.6 and follow the same steps of the proof for $\delta=1$ deletion while considering $t$ cases instead of $k / \log k$. Some of the steps will be skipped for the sake of brevity.

$$
\begin{align*}
\operatorname{Pr}\left(F \mid p_{1}, \ldots, p_{\delta}\right) & \leq \operatorname{Pr}\left(\bigcup_{j=2}^{t}\left\{\mathcal{Y}_{\boldsymbol{j}} \in \mathrm{A}, \mathcal{Y}_{\boldsymbol{j}} \neq \mathcal{Y}_{\mathbf{1}}\right\} \mid p_{1}, \ldots, p_{\delta}\right)  \tag{2.14}\\
& \leq \sum_{j=2}^{t} \operatorname{Pr}\left(\mathcal{Y}_{\boldsymbol{j}} \in \mathrm{A} \mid p_{1}, \ldots, p_{\delta}\right)  \tag{2.15}\\
& \leq \sum_{j=2}^{t} \sum_{\mathbf{Y} \in A} \frac{h(\delta)}{q^{\frac{k}{\log k}-\delta}}  \tag{2.16}\\
& <\frac{t \cdot h(\delta)}{q^{c-\delta}} \tag{2.17}
\end{align*}
$$

Furthermore,

$$
\begin{align*}
\operatorname{Pr}(F) & =\sum_{p_{1}, \ldots, p_{\delta} \in G F(q)} \operatorname{Pr}\left(F \mid p_{1}, \ldots, p_{\delta}\right) \operatorname{Pr}\left(p_{1}, \ldots, p_{\delta}\right)  \tag{2.18}\\
& <\sum_{p_{1}, \ldots, p_{\delta} \in G F(q)} \frac{t \times h(\delta)}{q^{c-\delta}} \operatorname{Pr}\left(p_{1}, \ldots, p_{\delta}\right)  \tag{2.19}\\
& =\frac{t \cdot h(\delta)}{q^{c-\delta}} . \tag{2.20}
\end{align*}
$$

Therefore,

$$
\begin{equation*}
\operatorname{Pr}(F)=\mathcal{O}\left(\frac{1}{k^{c-2 \delta} \log ^{\delta} k}\right) \tag{2.21}
\end{equation*}
$$

(2.16) follows from Claim 2.6. (2.19) follows from (2.17). (2.21) follows from (2.3), (2.20) and the fact that $h(\delta)$ is constant (independent of $k$ ) for a constant $\delta$.

### 2.6.5 Proof of Claim 2.5

We focus on case $i$ ( $i$ fixed) that assumes that the deletion has occurred in block $i$. We observe the output $\mathcal{Y}_{i}$ of the decoder in step 3 of the decoding scheme for all possible input messages, for a fixed deletion position and a given parity $p_{1}$. Recall that $\mathcal{Y}_{\boldsymbol{i}}$ is a random variable taking values in $\mathrm{A}_{1}$. Claim 2.5 gives an upper bound on the probability mass function of $\boldsymbol{\mathcal { Y }}_{\boldsymbol{i}}$ for any $i$ and for a given $p_{1}$. We distinguish here between two cases. If case $i$ is correct, i.e., the assumption on the deletion position is correct, then $\mathcal{Y}_{\boldsymbol{i}}$ is always decoded correctly and it is uniformly distributed over $A_{1}$. If case $i$ is wrong, then $\mathcal{Y}_{\boldsymbol{i}}$ is not uniformly distributed over $A_{1}$ as illustrated in the next example.

Example 2.4. Let the length of the binary message $\mathbf{u}$ be $k=16$ and consider $\delta=1$ deletion. Let the first MDS parity $p_{1}$ be the sum of the $k / \log k=4$ message symbols. Consider the previously defined set $\mathrm{A}_{1}$ with $p_{1}=0$. Consider the messages,

$$
\begin{aligned}
& \mathbf{U}_{\mathbf{1}}=(0,0,0,0) \in \mathrm{A}_{1}, \\
& \mathbf{U}_{\mathbf{2}}=(\alpha, 0,0, \alpha) \in \mathrm{A}_{1}
\end{aligned}
$$

For the sake of simplicity, we skip the last encoding step (Block IV, Fig. 2.1), and assume that $p_{1}$ is recovered at the decoder. Therefore, the corresponding codewords to be transmitted are

$$
\begin{aligned}
& \mathbf{x}_{\mathbf{1}}=00 \underline{o} 00000000000000000 \\
& \mathbf{x}_{\mathbf{2}}=00 \underline{1} 00000000000100000
\end{aligned}
$$

Now, assume that the $3^{\text {rd }}$ bit of $\mathbf{x}_{\mathbf{1}}$ and $\mathbf{x}_{\mathbf{2}}$ was deleted, and case 4 (wrong case) is considered. It easy to verify that in this case, for both codewords, the $q$-ary output of the decoder will be

$$
\mathcal{Y}_{4}=(0,0,0,0) \in \mathrm{A}_{1}
$$

This shows that there exists a wrong case $i$, where the same output can be obtained for two different inputs and a fixed deletion position. Thereby, the distribution of $\mathcal{Y}_{\boldsymbol{i}}$ over $\mathrm{A}_{1}$ is not uniform.

The previous example suggests that to find the bound in Claim 2.5, we need to determine the maximum number of different inputs that can generate the same output for an arbitrary fixed deletion position and a given parity $p_{1}$. We call this number $\gamma$. Once we obtain $\gamma$ we can write

$$
\begin{equation*}
\operatorname{Pr}\left(\mathcal{Y}_{i}=\mathbf{Y} \mid \mathcal{D}=d, \mathcal{P}_{1}=p_{1}\right) \leq \frac{\gamma}{\left|\mathrm{A}_{1}\right|}=\frac{\gamma}{q^{\frac{k}{\log k}-1}} \tag{2.22}
\end{equation*}
$$

where $\mathcal{D} \in\{1, \ldots, n\}$ is the random variable representing the position of the deleted bit. We will explain our approach for determining $\gamma$ by going through an example for $k=16$ that can be easily generalized for any $k$. We denote by $b_{z} \in G F(2), z=1,2, \ldots, k$, the bit of the message $\mathbf{u}$ in position $z$.

Example 2.5. Let $k=16$ and $\delta=1$. Consider the binary message $\mathbf{u}$ given by

$$
\mathbf{u}=b_{1} b_{2} b_{3} b_{4} b_{5} b_{6} b_{7} b_{8} b_{9} b_{10} b_{11} b_{12} b_{13} b_{14} b_{15} b_{16}
$$

The extension field used here has a primitive element $\alpha$, with $\alpha^{4}=\alpha+1$. Assume that $b_{3}$ was deleted and case 4 is considered. Hence, the binary string received at the decoder is chunked as follows

$$
\underbrace{b_{1} b_{2} b_{4} b_{5}}_{S_{1}} \underbrace{b_{6} b_{7} b_{8} b_{9}}_{S_{2}} \underbrace{b_{10} b_{11} b_{12} b_{13}}_{S_{3}} \underbrace{b_{14} b_{15} b_{16}}_{\mathcal{E}}
$$

where the erasure is denoted by $\mathcal{E}$, and $S_{1}, S_{2}$ and $S_{3}$ are the first 3 symbols of $\mathcal{Y}_{4}$ given by

$$
\begin{aligned}
& S_{1}=\alpha^{3} b_{1}+\alpha^{2} b_{2}+\alpha b_{4}+b_{5} \in G F(16), \\
& S_{2}=\alpha^{3} b_{6}+\alpha^{2} b_{7}+\alpha b_{8}+b_{9} \in G F(16) \\
& S_{3}=\alpha^{3} b_{10}+\alpha^{2} b_{11}+\alpha b_{12}+b_{13} \in G F(16)
\end{aligned}
$$

The fourth symbol $S_{4}$ of $\mathcal{Y}_{4}$ is to be determined by erasure decoding. Suppose that there exists another message $\mathbf{u}^{\prime} \neq \mathbf{u}$ such that $\mathbf{u}$ and $\mathbf{u}^{\prime}$ lead to the same decoded string $\mathcal{Y}_{\mathbf{4}}=\left(S_{1}, S_{2}, S_{3}, S_{4}\right)$. Since these two messages generate the same values of $S_{1}, S_{2}$ and $S_{3}$, then they should have the same values for the following bits

$$
b_{1} b_{2} b_{4} b_{5} b_{6} b_{7} b_{8} b_{9} b_{10} b_{11} b_{12} b_{13}
$$

We refer to these $k-\log k=12$ bits by "fixed" bits. The only bits that can be different in $\mathbf{u}$ and $\mathbf{u}^{\prime}$ are $b_{14}, b_{15}$ and $b_{16}$ which correspond to the erasure, and the deleted bit $b_{3}$. We refer to these $\log k=4$ bits by "free" bits. Although these "free" bits can be different in $\mathbf{u}$ and $\mathbf{u}^{\prime}$, they are constrained by the fact that the first parity in their corresponding $q$-ary codewords $\mathbf{X}$ and $\mathbf{X}^{\prime}$ should have the same value $p_{1}$. Next, we express this constraint by a linear equation in $G F(16)$. Without loss of generality, we assume that $p_{1} \in G F(16)$ is the sum of the $k / \log k=4$ message symbols. Hence, $p_{1}$ is given by
$p_{1}=\alpha^{3}\left(b_{1}+b_{5}+b_{9}+b_{13}\right)+\alpha^{2}\left(b_{2}+b_{6}+b_{10}+b_{14}\right)+\alpha\left(b_{3}+b_{7}+b_{11}+b_{15}\right)+\left(b_{4}+b_{8}+b_{12}+b_{16}\right)$.

Rewriting the previous equation by having the "free" bits on the LHS and the "fixed" bits and $p_{1}$ on the RHS we get

$$
\begin{equation*}
\alpha b_{3}+\alpha^{2} b_{14}+\alpha b_{15}+b_{16}=p^{\prime} \tag{2.23}
\end{equation*}
$$

where $p^{\prime} \in G F(16)$ and is given by $p^{\prime}=p_{1}+\alpha^{3}\left(b_{1}+b_{5}+b_{9}+b_{13}\right)+\alpha^{2}\left(b_{2}+b_{6}+b_{10}\right)+$ $\alpha\left(b_{7}+b_{11}\right)+\left(b_{4}+b_{8}+b_{12}\right)$. The previous equation can be written as the following linear equation in $G F(16)$,

$$
\begin{equation*}
0 \alpha^{3}+b_{14} \alpha^{2}+\left(b_{3}+b_{15}\right) \alpha+b_{16}=p^{\prime} \tag{2.24}
\end{equation*}
$$

Now, to determine $\gamma$, we count the number of solutions of (2.24). If the unknowns in (2.24) were symbols in $G F(16)$, then the solutions of (2.24) would span an affine subspace of size $q^{k / \log k-1}=16^{3}$. However, the unknowns in (2.24) are binary, so we show next that it has at most 2 solutions. Let

$$
\begin{equation*}
a_{3} \alpha^{3}+a_{2} \alpha^{2}+a_{1} \alpha+a_{0}=p^{\prime} \tag{2.25}
\end{equation*}
$$

be the polynomial representation of $p^{\prime}$ in $G F(16)$ where $\left(a_{3}, a_{2}, a_{1}, a_{0}\right) \in G F(2)^{4}$. Every element in $G F(16)$ has a unique polynomial representation of degree at most 3. Comparing (2.24) and (2.25), we obtain the following system of equations

$$
\left\{\begin{array}{cl}
b_{16} & =a_{0} \\
b_{3}+b_{15} & =a_{1} \\
b_{14} & =a_{2} \\
0 & =a_{3}
\end{array}\right.
$$

If $a_{3} \neq 0$, then (2.24) has no solution. If $a_{3}=0$, then (2.24) has 2 solutions because $b_{3}+b_{15}=a_{1}$ has 2 solutions. Therefore, (2.24) has at most 2 solutions, i.e., $\gamma \leq 2$.

The analysis in Example 2.5 can be directly generalized for messages of any length $k$. In general, the analysis yields $\log k$ "free" bits and $k-\log k$ "fixed" bits. Now, we generalize (2.24) and show that $\gamma \leq 2$ for any $k$. Without loss of generality, we assume that $p_{1} \in G F(q)$ is the
sum of the $k / \log k$ symbols of the $q$-ary message $\mathbf{U}$. Consider a wrong case $i$ that assumes that the deletion has occurred in block $i$. Let $d_{j}$ be a fixed bit position in block $j, j \neq i$, that represents the position of the deletion. Depending on whether the deletion occurred before or after block $i$, the generalization of (2.24) is given by one of the two following equations in $G F(q)$.

If $j<i$,

$$
\begin{equation*}
b_{d_{j}} \alpha^{w}+b_{\ell+1} \alpha^{m-1}+b_{\ell} \alpha^{m-2}+\ldots+b_{\ell+m}=p^{\prime \prime} \tag{2.26}
\end{equation*}
$$

If $j>i$,

$$
\begin{equation*}
b_{d_{j}} \alpha^{w}+b_{\ell} \alpha^{m}+b_{\ell+1} \alpha^{m-1}+\ldots+b_{\ell+m-1} \alpha=p^{\prime \prime} \tag{2.27}
\end{equation*}
$$

where $\ell=(i-1) \log k+1, m=\log k-1, w=j \log k-b_{j}$ and $p^{\prime \prime} \in G F(q)$ (the generalization of $p^{\prime}$ in Example 2.5) is the sum of $p_{1}$ and the part corresponding to the "fixed" bits. Suppose that $j<i$. Note that $1 \leq w \leq m$, so (2.26) can be written as

$$
\begin{equation*}
b_{\ell+1} \alpha^{m-1}+\ldots+\left(b_{d_{j}}+b_{\ell+o}\right) \alpha^{w}+\ldots+b_{\ell+m}=p^{\prime \prime} \tag{2.28}
\end{equation*}
$$

where $o$ is an integer such that $1 \leq o \leq m$. Hence, by the same reasoning used in Example 2.5 we can conclude that (2.28) has at most 2 solutions. The same reasoning applies for (2.27), where $j>i$. Therefore, $\gamma \leq 2$ and from (3.14) we have

$$
\begin{equation*}
\operatorname{Pr}\left(\mathcal{Y}_{i}=\mathbf{Y} \mid \mathcal{D}=d, \mathcal{P}_{1}=p_{1}\right) \leq \frac{2}{q^{\frac{k}{\log k}-1}} \tag{2.29}
\end{equation*}
$$

The bound in (3.20) holds for arbitrary $d$. Therefore, the upper bound on the probability of decoding failure in (2.3) holds for any deletion position picked independently of the codeword. Moreover, for any given random distribution on $\mathcal{D}$ (like the uniform distribution for example), we can apply the total law of probability with respect to $\mathcal{D}$ and use the result from (3.20) to get

$$
\operatorname{Pr}\left(\mathcal{Y}_{\boldsymbol{i}}=\mathbf{Y} \mid \mathcal{P}_{1}=p_{1}\right) \leq \frac{2}{q^{\frac{k}{\log k}-1}}
$$

### 2.7 Trade-offs

As previously mentioned, the first encoding step in GC codes (Block I, Fig. 2.1) consists of chunking the message into blocks of length $\log k$ bits. In this section, we generalize the results in Theorem 2.1 by considering chunks of arbitrary length $\ell$ bits $(\ell \leq k)^{7}$, instead of $\log k$ bits. We show that if $\ell=\Omega(\log k)$, then GC codes have an asymptotically vanishing probability of decoding failure. This generalization allows us to demonstrate two trade-offs achieved by GC codes, based on the code properties in Theorem ??.

Theorem 2.7. Guess 8 Check ( $G C$ ) codes can correct in polynomial time up to a constant number of $\delta$ deletions. Let $c>\delta$ be an integer. The code has the following properties:

1. Redundancy: $n-k=c(\delta+1) \ell$ bits.
2. Encoding complexity is $\mathcal{O}(k \ell)$, and decoding complexity is $\mathcal{O}\left(\frac{k^{\delta+1}}{\ell^{\delta-1}}\right)$.
3. For any $\delta$ deletion positions chosen independently of the codeword, the probability that the decoding fails for a uniform iid message is: $\operatorname{Pr}(F)=\mathcal{O}\left(\frac{(k / \ell)^{\delta}}{2^{\ell(c-\delta)}}\right)$.

Proof. See Appendix A.

The code properties in Theorem 2.7 enable two trade-offs for GC codes:

1) Trade-off $A$ : By increasing $\ell$ for a fixed $k$ and $c$, we observe from Theorem 2.7 that the redundancy increases linearly while the decoding complexity decreases as a polynomial of degree $\delta$. Moreover, increasing $\ell$ also decreases the probability of decoding failure as a polynomial of degree $\delta$. Therefore, trade-off A shows that by paying a linear price in terms of redundancy, we can simultaneously gain a degree $\delta$ polynomial improvement in both decoding complexity and probability of decoding failure.
2) Trade-off B: By increasing $c$ for a fixed $k$ and $\ell$, we observe from Theorem 2.7 that the redundancy increases linearly while the probability of decoding failure decreases exponentially. Here, the order of complexity is not affected by $c$. Therefore, trade-off B shows that by paying

[^9]a linear price in terms of redundancy, we can gain an exponential improvement in probability of decoding failure, without affecting the order of complexity. This trade-off is of particular importance in models which allow feedback, where asking for additional parities (increasing $c$ ) will highly increase the probability of successful decoding.

### 2.8 Correcting $\delta$ Insertions

In this section we show that by modifying the decoding scheme of GC codes, and keeping the same encoding scheme, we can obtain codes that can correct $\delta$ insertions, instead of $\delta$ deletions. In fact, the resulting code properties for $\delta$ insertions (Theorem 2.8), are the same as that of $\delta$ deletions. Recall that $\ell$ (Section 2.7) is the code parameter representing the chunking length, i.e., the number of bits in a single block. For correcting $\delta$ insertions, we keep the same encoding scheme as in Fig. 2.1, and for decoding we only modify the following: (i) Consider the assumption that a certain block $B$ is affected by $\delta^{\prime}$ insertions, then while decoding we chunk a length of $\ell+\delta^{\prime}$ at the position of block $B$ (compared to chunking $\ell-\delta^{\prime}$ bits when decoding deletions);
(ii) The blocks assumed to be affected by insertions are considered to be erased (same as the deletions problem), but now for Criterion 2 (Definition 2.4) we check if the decoded erasure is a subsequence of the chunked super-block (of length $\ell+\delta^{\prime}$ ). The rest of the details in the decoding scheme stay the same.

Theorem 2.8. Guess $\xi^{3}$ Check $(G C)$ codes can correct in polynomial time up to a constant number of $\delta$ insertions. Let $c>\delta$ be an integer. The code has the following properties:

1. Redundancy: $n-k=c(\delta+1) \ell$ bits.
2. Encoding complexity is $\mathcal{O}(k \ell)$, and decoding complexity is $\mathcal{O}\left(\frac{k^{\delta+1}}{\ell^{\delta-1}}\right)$.
3. For any $\delta$ insertion positions chosen independently of the codeword, the probability that the decoding fails for a uniform iid message is: $\operatorname{Pr}(F)=\mathcal{O}\left(\frac{(k / \ell)^{\delta}}{2^{\ell(c-\delta)}}\right)$.

We omit the proof of Theorem 2.8 because the same analysis applies as in the proof of Theorem 2.7. More specifically, the redundancy and the encoding complexity are the same as
in Theorem 2.7 because we use the same encoding scheme. The decoding complexity is also the same as in Theorem 2.7 because the total number of cases to be checked by the decoder is unchanged. The proof of the upper bound on the probability of decoding failure also applies similarly using the same techniques.

### 2.9 Simulation Results

We simulated the decoding of GC codes and compared the obtained probability of decoding failure to the upper bound in Theorem 2.1. We tested the code for messages of length $k=256,512$ and 1024 bits, and for $\delta=2,3$ and 4 deletions. To guarantee an asymptotically vanishing prob-

| Config. | $\delta$ |  |  |  |  |  |  | 3 |  | 4 |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | 2 |  | $\operatorname{Pr}(F)$ | $R$ | $\operatorname{Pr}(F)$ | $R$ |  |  |  |  |  |
| $\operatorname{Pr}(F)$ |  |  |  |  |  |  |  |  |  |  |  |
| $k$ | $R$ | $1.3 e^{-3}$ | 0.67 | $4.0 e^{-4}$ | 0.56 | 0 |  |  |  |  |  |
| 256 | 0.78 | 10.69 | 0 |  |  |  |  |  |  |  |  |
| 512 | 0.86 | $3.0 e^{-4}$ | 0.78 | 0 | 0.69 | 0 |  |  |  |  |  |
| 1024 | 0.92 | $2.0 e^{-4}$ | 0.86 | 0 | 0.80 | 0 |  |  |  |  |  |

Table 2.1: The table shows the code rate $R=k / n$ and the probability of decoding failure $\operatorname{Pr}(F)$ of GC codes for different message lengths $k$ and different number of deletions $\delta$. The results shown are for $c=\delta+1$ and $\ell=\log k$. The results of $\operatorname{Pr}(F)$ are averaged over 10000 runs of simulations. In each run, a message $\mathbf{u}$ chosen uniformly at random is encoded into the codeword $\mathbf{x} . \delta$ bits are then deleted uniformly at random from $\mathbf{x}$, and the resulting string is decoded.
ability of decoding failure, the upper bound in Theorem 2.1 requires that $c>2 \delta$. Therefore, we make a distinction between two regimes, (i) $\delta<c<2 \delta$ : Here, the theoretical upper bound is trivial. Table 3.1 gives the results for $c=\delta+1$ with the highest probability of decoding failure observed in our simulations being of the order of $10^{-3}$. This indicates that GC codes can decode correctly with high probability in this regime, although not reflected in the upper bound; (ii) $c>2 \delta$ : The upper bound is of the order of $10^{-5}$ for $k=1024, \delta=2$, and $c=2 \delta+1$. In the simulations no decoding failure was detected within 10000 runs for $\delta+2 \leq c \leq 2 \delta+1$. In general, the simulations show that GC codes perform better than what the upper bound indicates. This is due to the fact that the effect of Criterion 2 (Definition 2.4) is not taken into account when deriving the upper bound in Theorem 2.1. These simulations were performed
on a personal computer and the programming code was not optimized. The average decoding time ${ }^{8}$ is in the order of milliseconds for $(k=1024, \delta=2)$, order of seconds for $(k=1024, \delta=3)$, and order of minutes for $(k=1024, \delta=4)$. Going beyond these values of $k$ and $\delta$ will largely increase the running time due to the number of cases to be tested by the decoder. However, for the file synchronization application in which we are interested (see next section) the values of $k$ and $\delta$ are relatively small and decoding can be practical.

### 2.10 Application to File Synchronization

In this section, we describe how GC codes can be used to construct interactive protocols for file synchronization. We consider the model where two nodes (servers) have copies of the same file but one is obtained from the other by deleting $d$ bits. These nodes communicate interactively over a noiseless link to synchronize the file affected by deletions. Some of the most recent work on synchronization can be found in $[1-3,5,6]$. In this section, we modify the synchronization algorithm by Venkataramanan et al. [1,5], and study the improvement that can be achieved by including GC codes as a black box inside the algorithm. The key idea in [1,5] is to use center bits to divide a large string, affected by $d$ deletions, into shorter segments, such that each segment is affected by one deletion at most. Then, use VT codes to correct these shorter segments. Now, consider a similar algorithm where the large string is divided such that the shorter segments are affected by $\delta(1<\delta \ll d)$ or fewer deletions. Then, use GC codes to correct the segments affected by more than one deletion ${ }^{9}$. We set $c=\delta+1$ and $\ell=\log k$, and if the decoding fails for a certain segment, we send one extra MDS parity at a time within the next communication round until the decoding is successful. By implementing this algorithm, the gain we get is two folds: (i) reduction in the number of communication rounds; (ii) reduction in the total communication cost. We performed simulations for $\delta=2$ on files of size 1 Mb , for different numbers of deletions $d$. The results are illustrated in Table 2.2. We refer to the original scheme

[^10]in $[1,5]$ by Sync-VT, and to the modified version by Sync-GC. The savings for $\delta=2$ are roughly $43 \%$ to $73 \%$ in number of rounds, and $5 \%$ to $14 \%$ in total communication cost.

|  | Number of rounds |  | Total communication cost |  |
| :---: | :---: | :---: | :---: | :---: |
| $d$ | Sync-VT | Sync-GC | Sync-VT | Sync-GC |
| 100 | 14.52 | 10.15 | 5145.29 | 4900.88 |
| 150 | 16.45 | 10.48 | 7735.32 | 7199.20 |
| 200 | 17.97 | 10.88 | 10240.60 | 9332.68 |
| 250 | 18.93 | 11.33 | 12785.20 | 11415.90 |
| 300 | 20.29 | 11.70 | 15318.20 | 13397.80 |

Table 2.2: Results are averaged over 1000 runs. In each run, a string of size 1 Mb is chosen uniformly at random, and the file to be synchronized is obtained by deleting $d$ bits from it uniformly at random. The total communication cost is expressed in bits. The number of center bits used is 25 .

Note that the interactive algorithm in $[1,5]$ also deals with the general file synchronization problem where the edited file is affected by both deletions and insertions. There, the string is divided into shorter segments such that each segment is either: (i) not affected by any deletions/insertions; or (ii) affected by only one deletion; or (iii) affected by only one insertion. Then, VT codes are used to correct the short segments. GC codes can also be used in a similar manner when both deletions and insertions are involved. In this case, the string would be divided such that the shorter segments are affected by: (i) $\delta$ or fewer deletions; or (ii) $\delta$ or fewer insertions.

### 2.11 List Decoding

In this section, we are interested in the problem of designing efficient codes for list decoding of deletions. A list decoder returns a list of candidate strings which is guaranteed to contain the actual codeword. The idea of list decoding was first introduced in the 1960s by Elias [63] and Wozencraft [64]. The main goal when studying list decoders is to find explicit codes that can return a small list of candidate strings in polynomial time ${ }^{10}$. The size of the list gives a

[^11]lower bound on the time complexity of the list decoder. For instance, if the size of the list is superpolynomial, then polynomial time list decoding cannot be achieved. List decoding has been studied for various classes of error correcting codes, such as Reed-Solomon codes [65], Reed-Muller codes [66], and Polar codes [67]. However, the problem of finding list decoders for deletions has not received much attention in the literature. In [25], Guruswami and Wang proved the existence of codes that can list-decode a constant fraction of deletions given by $n\left(\frac{1}{2}-\varepsilon\right)$, where $0<\varepsilon<\frac{1}{2}$ and $n$ is the length of the codeword. In the regime considered in [25], the codes have low rate of the order of $\varepsilon^{3}$. Recently in [68], Wachter-Zeh derived an upper bound on the list size for decoding deletions and insertions. An explicit list decoding algorithm that is based on VT codes was also proposed in [68].

Here, we focus on the case where the codeword $\mathbf{x}$ is affected by a constant number of deletions $\delta$, resulting in a string $\mathbf{y}$ of length $n-\delta$. A simple list decoder in this case is one that returns all binary strings that have a Levenshtein distance $\delta$ from $\mathbf{y}$, i.e., all supersequences of $\mathbf{y}$ of length $n$. This list decoder does not require any redundancy, and its resulting list size is exactly $\sum_{i=0}^{\delta}\binom{n}{i}$ [69]. Hence, for a constant number of deletions $\delta$, the list size is $\mathcal{O}\left(n^{\delta}\right)$ (polynomial function of $n$ of degree $\delta$ ). In [68], this list size was reduced to $\mathcal{O}\left(n^{\delta-1}\right)$ by using VT codes. The idea in [68] is to first generate all binary strings that have a Levenshtein distance $\delta-1$ from $\mathbf{y}$, and then decode these strings using VT codes. Note that this reduction in the list size comes at the expense of adding a logarithmic redundancy that is introduced by VT codes.

In this section, we study the list decoding performance of GC codes. To this end, we quantify the value of the list size obtained by the GC decoder by studying the average and the maximum size of the list. Through our theoretical and simulation results, we show that for a constant number of deletions $\delta$, GC codes with logarithmic redundancy can return a small list of candidate strings in polynomial time. Namely, our contributions are the following.

Theoretical results: Our theoretical results show that the average size of the list approaches 1 asymptotically in $k$ for a uniform iid message and any deletion pattern. These results demonstrate that in the average case, lists of size strictly greater than one occur with low probability.

Simulation results: We provide numerical simulations on the list decoding performance of GC codes for values of $k$ up to 1024 bits, and values of $\delta$ up to 3 deletions. The average list size recorded in these simulations is very close to 1 (less than 2 ). Whereas, the maximum list size detected within the performed simulations is 3 . Consequently, we conjecture that for a constant number of deletions, the maximum size of the list for any message and any deletion pattern, is upper bounded by a constant that is independent of $k$.

Comparison to [68]: Our theoretical results improve on the list decoder presented in [68], whose maximum list size is theoretically $\mathcal{O}\left(n^{\delta-1}\right)$, i.e., upper bounded by a function that grows polynomially in length of the codeword $n$ for a constant number of deletions $\delta$. Furthermore, we also provide a numerical comparison to [68] which shows that the maximum list size of list decoder in [68] grows with $n$ and is much larger than that of GC codes.

### 2.11.1 Preliminaries

Recall that the GC decoder described in Section 2.5 generates

$$
\begin{equation*}
t=\binom{\lceil k / \ell\rceil+\delta-1}{\delta}=\mathcal{O}\left(\frac{k^{\delta}}{\ell^{\delta}}\right) \tag{2.30}
\end{equation*}
$$

guesses, corresponding to the possible locations of the $\delta$ deletion among the $\lceil k / \ell\rceil$ blocks. This guessing phase results in an initial list of at most ${ }^{11} t$ decoded strings. The decoder then checks whether each decoded string in the initial list is a valid guess or not, and removes the invalid ones. A guess is considered is to be valid if the decoded string is consistent with the additional parities; and its Levenshtein distance from the received string $\mathbf{y}$ is exactly $\delta$. At the end of this checking phase, the GC decoder is left with a smaller final list of candidate strings.

Proposition 2.9. The final list returned by the $G C$ decoder is guaranteed to contain the actual codeword $\mathbf{x}$, and all the strings in this list have a Levenshtein distance $\delta$ from the received string $\mathbf{y}$.

[^12]Proof. The decoder goes over all possible deletion patterns, so the actual deletion pattern is guaranteed to be considered in one of the $t$ guesses. Furthermore, the parities are recovered with zero-error since they are protected by a $(\delta+1)$ repetition code. Therefore, the decoding of $\mathbf{y}$ for the correct guess will result in the actual codeword $\mathbf{x}$. Also, the fact that all the strings in this list have a Levenshtein distance $\delta$ from $\mathbf{y}$, follows directly from the last decoding step described in Section 2.5.

Since the size of the initial list is at most $t$, and $t$ is upper bounded by a polynomial function of $k$ given by (2.30), then the initial list size is at most polynomial in $k$. We are interested in studying the size of the final list obtained by the GC decoder. The list size is a deterministic function of the codeword $\mathbf{x}$ and the deletion pattern $\mathbf{d}$, and hence can be represented by $L(\mathbf{x}, \mathbf{d})$. Since the codeword $\mathbf{x}$ is a deterministic function of the message $\mathbf{u}$, an equivalent definition of the list size is $L(\mathbf{u}, \mathbf{d})$. Henceforth, we drop the $\mathbf{d}$ argument and use $L(\mathbf{u})$ to refer to the maximum list size over all possible deletion patterns for a message $\mathbf{u}$ of length $k$ bits, i.e.,

$$
\begin{equation*}
L(\mathbf{u}) \triangleq \max _{\mathbf{d}} L(\mathbf{u}, \mathbf{d}) \tag{2.31}
\end{equation*}
$$

Based on the decoding steps of GC codes, we have $L(\mathbf{u}) \in\{1, \ldots, t\}$. To quantify the size of the final list we define the following quantities:

1. The average value of the list size, defined by

$$
\begin{equation*}
L_{a v} \triangleq \mathbb{E}(L(\mathbf{u}))=\sum_{l=1}^{t} l \cdot \operatorname{Pr}(L(\mathbf{u})=l) \tag{2.32}
\end{equation*}
$$

for a uniform iid message $\mathbf{u}$ of length $k$ bits.
2. The maximum value of the list size, defined by

$$
\begin{equation*}
L_{\max } \triangleq \max _{\mathbf{u}} L(\mathbf{u}) \tag{2.33}
\end{equation*}
$$

for any message $\mathbf{u}$ of length $k$ bits.

The definition of $L(\mathbf{u})$ in (2.31) implies that the average $L_{a v}$ defined in (2.32), and the maximum $L_{\max }$ defined in (2.33), are maximized over all possible deletion patterns.

### 2.11.2 Theoretical Results

Recall that $\ell$ represents the chunking length for GC codes, and that the number of deletions $\delta$ and the number of parity symbols $c$ are constants, i.e., independent of $k$. Theorem 2.10 gives an upper bound on the average list size $L_{a v}$ defined in (2.32), in terms of the GC code parameters.

Theorem 2.10 (Average list size). For a uniform iid message of length $k$ bits, and any deletion pattern $\left(d_{1}, d_{2}, \ldots, d_{\delta}\right)$, the average list size $L_{\text {av }}$ obtained by the Guess \& Check (GC) decoder satisfies

$$
1 \leq L_{a v} \leq 1+\mathcal{O}\left(\frac{(k / \ell)^{2 \delta}}{2^{\ell(c-\delta)}}\right)
$$

Proof. For brevity, we use $L$ instead of $L(\mathbf{u})$ throughout the proof. As previously mentioned, if $L \geq 2$, then we say that the decoder failed to decode uniquely. The probability of failure for a uniform iid binary message $\mathbf{u}$ of length $k$ bits, and any deletion pattern $\mathbf{d}=\left(d_{1}, \ldots, d_{\delta}\right)$, is upper bounded by the expression given in Theorem 2.7. Recall that $L \in\{1, \ldots, t\}$, where $t$ is the total number of cases checked by the GC decoder, given by (2.30). The lower bound on $L_{a v}$ follows directly from the fact that $L \geq 1$, and hence

$$
\begin{equation*}
L_{a v} \geq 1 \tag{2.34}
\end{equation*}
$$

To upper bound $L_{a a v}$, we write the following.

$$
\begin{align*}
L_{a v} & =\sum_{l=1}^{\infty} \operatorname{Pr}(L \geq l)  \tag{2.35}\\
& =\sum_{l=1}^{t} \operatorname{Pr}(L \geq l)  \tag{2.36}\\
& =1+\sum_{l=2}^{t} \operatorname{Pr}(L \geq l)  \tag{2.37}\\
& \leq 1+t \operatorname{Pr}(L \geq 2)  \tag{2.38}\\
& =1+\mathcal{O}\left(\frac{k^{\delta}}{\ell^{\delta}} \cdot \frac{k^{\delta}}{\ell^{\delta} 2^{\ell(c-\delta)}}\right)  \tag{2.39}\\
& =1+\mathcal{O}\left(\frac{k^{2 \delta}}{\ell^{2 \delta} 2^{\ell(c-\delta)}}\right) . \tag{2.40}
\end{align*}
$$

Equations (2.35) to (2.37) follow since $L \in\{1, \ldots, t\}$ is a positive integer-valued random variable. (2.39) follows from the fact that $\operatorname{Pr}(L \geq 2)=\operatorname{Pr}(F)$, and from (2.30) and Theorem 2.7.

The next result follows from Theorem 2.10 and shows that for appropriate choices of the GC code parameters, $L_{a v}$ approaches one asymptotically in the length of the message $k$.

Corollary 2.11. For choices of the $G C$ code parameters that satisfy $\ell=\Omega(\log k)$ and $c \geq 3 \delta$, the average list size satisifies

$$
\lim _{k \rightarrow+\infty} L_{a v}=1
$$

Proof. Let $\varepsilon>0$, such that $\varepsilon=\mathcal{O}\left(k^{2 \delta} / \ell^{2 \delta} 2^{\ell(c-\delta)}\right)$. Based on Theorem 2.10 we have

$$
\begin{equation*}
1 \leq L_{a v} \leq 1+\varepsilon . \tag{2.41}
\end{equation*}
$$

To prove that $\lim _{k \rightarrow+\infty} L_{a v}=1$, we derive conditions on the code parameters under which $\varepsilon$ is guaranteed to vanish asymptotically in $k$. $\varepsilon$ goes to zero as $k$ approaches infinity if the denominator in its mathematical expression converges to zero faster than the numerator. It is easy to verify that this holds when $\ell=\Omega(\log k)$. Let $\ell=\Omega(\log k)$ be the first condition. Then,
we have

$$
\begin{equation*}
\varepsilon=\mathcal{O}\left(\frac{1}{k^{c-3 \delta}(\log k)^{2 \delta}}\right) \tag{2.42}
\end{equation*}
$$

Hence, we obtain a second condition that $c \geq 3 \delta$. Therefore, for $\ell=\Omega(\log k)$ and $c \geq 3 \delta, L_{a v}$ approaches 1 as the message length $k$ (or equivalently the block length $n$ ) goes to infinity.

Recall that the redundancy of GC codes is $n-k=c(\delta+1) \ell$. Let $\ell=\log k$ be the chunking length used for encoding. In this case, the redundancy is $c(\delta+1) \log k$, i.e., logarithmic in $k$. It follows from Corollary 2.11 that a logarithmic redundancy is sufficient for GC codes so that $\lim _{k \rightarrow \infty} L_{a v}=1$. It is easy to verify that a logarithmic redundancy corresponds to a code rate $R=\frac{k}{n}$ that is asymptotically optimal in $n$ (rate approaches one as $n$ goes to infinity). Therefore, GC codes can achieve the list decoding properties given in Theorem 2.10 with a logarithmic redundancy and an asymptotically optimal code rate.

### 2.11.3 Numerical Results

In this section we present simulation results on the average and maximum list size obtained by GC codes. We also compare the list decoding performance of GC codes to that of the codes presented in [68].

| Config. | $\delta$ |  |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
|  | 1 |  | 2 |  | 3 |  |
| $k$ | $L_{a v}$ | $L_{\text {max }}$ | $L_{a v}$ | $L_{\text {max }}$ | $L_{a v}$ | $L_{\text {max }}$ |
| 32 | 1.0183 | 2 | 1.0151 | 3 | 1.0061 | 3 |
| 64 | 1.0110 | 2 | 1.0061 | 3 | 1.0043 | 3 |
| 128 | 1.0069 | 2 | 1.0029 | 2 | 1.0020 | 2 |
| 256 | 1.0035 | 2 | 1.0020 | 2 | 1.0007 | 2 |
| 512 | 1.0021 | 2 | 1.0010 | 2 | 1.0005 | 2 |
| 1024 | 1.0007 | 2 | 1.0005 | 2 | 1.0002 | 2 |

Table 2.3: The table shows the average list size $L_{a v}$ and the maximum list size $L_{\max }$ obtained by the GC decoder for different message lengths $k$ and different number of deletions $\delta$. The results shown are for $c=\delta+1$ and $\ell=\log k$. The results of $L_{a v}$ and $L_{\max }$ were recorded over 10000 runs of simulations. In each run, a message $\mathbf{u}$ chosen uniformly at random is encoded into the codeword $\mathbf{x} . \delta$ bits are then deleted from $\mathbf{x}$ based on a uniformly distributed deletion pattern $\mathbf{d}$, and the resulting string is decoded.

We performed numerical simulations on the average list size $L_{a v}$ and the maximum list size $L_{\max }$ for $k=32,64,128,256,512$ and 1024 bits, and for $\delta=1,2$ and 3 deletions. The empirical results are shown in Table II.

The results show that: (i) the average list size $L_{a v}$ is very close to one and its value approaches one further as $k$ increases; and (ii) the maximum list size $L_{\max }$ recorded is 3 and $L_{\max }$ does not increase with $k$, for $k=32,64, \ldots, 1024$. Note that the redundancy used for these simulations is $n-k=c(\delta+1) \log k$ with $c=\delta+1$. This redundancy is much smaller than the one suggested by Theorem 2.10. This is due to the fact that GC codes perform better than what the theoretical bounds indicate, which was discussed in Section 2.9.


Figure 2.2: The figure shows the maximum list size $L_{\max }$ obtained by GC codes and the codes in [1] for $\delta=2$ deletions and different message lengths. The GC code parameters are set to $c=\delta+1$ and $\ell=\log k$. The results are obtained over 10000 runs of simulations. In each run, the message and the deletion pattern are chosen independently and uniformly at random.

In [68], a list decoder of $\delta$ deletions was presented that is based on VT codes [18]. Recall that VT codes can uniquely decode a single deletion. Consider a codeword that is affected by $\delta$ deletions, resulting in a string $\mathbf{y}$ of length $n-\delta$ bits. The main idea in [68] is to first generate all the supersequences of $\mathbf{y}$ of length $n-1$; and then apply the VT decoder on each supersequence. The decoding results in a list whose maximum size is theoretically $\mathcal{O}\left(n^{\delta-1}\right)$. Note that this size increases polynomially in $n$ for a constant $\delta$. We simulated the maximum list size of the list decoder in [68], and compared it to that of GC codes for $\delta=2$. The obtained results are shown in Fig. 2.2. The comparison shows that the maximum list size in [68] is larger and increases with
the message length $k$. Note that the two compared codes have the same order of redundancy (logarithmic in $n$ ) and the same order of decoding complexity (polynomial in $n$ for constant $\delta$ ).

We conclude this section with a conjecture. Based on the results obtained above, we conjecture that the maximum list size $L_{\max }$ defined in (2.33), is upper bounded by a constant.

Conjecture 2.12. For appropiate choices of the GC codes parameters, the maximum list size $L_{\text {max }}$, for any message of length $k \in\left\{k_{1}, k_{2}, k_{3}, \ldots\right\}$, and any deletion pattern $\left(d_{1}, \ldots, d_{\delta}\right)$, is upper bounded by a constant that is independent of $k$.

This conjecture is consistent with the empirical results in Table 2.3, where we gradually increase the message length from $k=32$ to $k=1024$ for multiple values of $\delta$, and observe that the maximum list size does not increase with $k$. Attempting to prove this conjecture is part of future work.

### 2.12 Conclusion

In this chapter, we introduced a new family of systematic codes, that we called Guess \& Check (GC) codes, that can correct multiple deletions (or insertions) with high probability. We provided deterministic polynomial time encoding and decoding schemes for these codes. We validated our theoretical results by numerical simulations. Moreover, we showed how these codes can be used in applications to remote file synchronization. We also studied the list decoding performance of GC codes and showed that these codes can return a small list of candidate strings in polynomial time.

In conclusion, we point out some open problems and possible directions of future work:

1. GC codes can correct $\delta$ deletions or $\delta$ insertions. Generalizing these constructions to deal with mixed deletions and insertions (indels) is a possible future direction.
2. Developing faster decoding algorithms for GC codes is also part of the future work.
3. From the proof of Theorem 2.1, it can be seen that bound on the probability of decoding
failure of GC codes holds if the deletion positions are chosen by an adversary that does not observe the codeword. It would be interesting to study the performance of GC codes under more powerful adversaries which can observe part of the codeword, while still allowing a vanishing probability of decoding failure.

## Chapter 3

## Codes for Localized Deletions

### 3.1 Introduction

In this chapter, we design efficient codes for the model where $\delta$ deletions are localized in a window of size at most $w$ bits. We also generalize these codes to the case of deletions that are localized within multiple windows. To the best of our knowledge, the only previous work on codes for localized deletions is the one by Schoeny et al. in [43] where the authors constructed codes for the particular case of a single window with $w=3$ or $w=4$. Moreover, the case of multiple windows has not been studied in the literature even for the simpler model of bursty deletions.

The codes that we present are based on our construction given in the previous chapter, where we proposed the Guess \& Check (GC) $\operatorname{codes}^{1}$ for the general problem of correcting $\delta>1$ unrestricted deletions. In this work, we exploit the localized nature of the deletions to modify the schemes in [71] and obtain codes that have an asymptotically optimal rate, and efficient encoding and decoding algorithms, for a number of localized deletions that grows with the block

[^13]length. Furthermore, for a uniform iid message, and for arbitrary deletion positions that are chosen independently of the codeword, these codes have an asymptotically vanishing probability of decoding failure ${ }^{2}$. Note that previous works on bursty deletions (e.g., [42-44]), have focused on constructing zero-error codes that can correct a constant number of consecutive deletions ( $\delta$ does not grow with $n$ ). However, it remains open to show whether the zero-error based constructions in [42-44] can result in practical polynomial time encoding and decoding schemes for the problem.

Namely, our contributions are the following:

1. We construct new explicit codes, that can correct, with high probability, $\delta \leq w$ deletions that are localized within a single window of size $w$ bits. In our construction, we allow $\delta$ and $w$ to grow with the block length.
2. These codes have a redundancy that is lower than the codes existing in the literature on bursts of deletions. Also, the code rate of these codes is asymptotically optimal for a sublinear number of localized deletions.
3. The complexity of the encoding algorithm is near-linear, and the complexity of decoding is quadratic.
4. We provide numerical simulations on the performance of these codes, and compare the simulation results to our theoretical bound on the probability of decoding failure.
5. We generalize our approach to obtain codes that can correct deletions that are localized within multiple windows in the codeword.

The chapter is organized as follows. In Section 3.2, we formally present the model for localized deletions and introduce the basic notation and terminology used throughout the chapter. We state and discuss our main results in Section 3.3, and compare them to the most recent

[^14]works on unrestricted and bursty deletions. In Section 3.4, we provide an encoding and decoding example of GC codes for correcting deletions thats are localized within a single window. In Section 3.5, we describe in detail our encoding and decoding schemes and discuss the choices of the code parameters. The results of the numerical simulations are presented in Section 3.6. In Section 3.7, we show the generalization of our codes to the case of deletions that are localized within multiple windows. The proof of the main result of this chapter is given in Section 3.8.

### 3.2 Preliminaries

In this chapter, we consider the following models for deletions:

1. Bursts of fixed lengths: the deletions occur in $z \geq 1$ bursts, where each burst corresponds to exactly $\delta \geq 1$ consecutive deletions.
2. Bursts of variable lengths: the deletions occur in $z \geq 1$ bursts, each of length at most $\delta \geq 1$ bits.
3. Localized deletions: the deletions are restricted to $z \geq 1$ windows. The size of each window is at most $w$ bits, and the number of deletions within each window is at most $\delta \leq w$. The bits that are deleted in a certain window are not necessarily consecutive.

Example 3.1 (Localized Deletions). Let $z=2, w=4$, and $\delta=3$. Let $\mathbf{x}$ be the transmitted string and $\mathbf{y}$ be the received string. The deletions (in red) are localized in $z=2$ windows.

$$
\begin{aligned}
& \mathbf{y}=1000010100100110 .
\end{aligned}
$$

Note that the two previously mentioned bursty models correspond to special cases of the localized model. Hence, a code that can correct localized deletions can also correct bursts of fixed or variable lengths.


Figure 3.1: Encoding block diagram of GC codes for correcting $\delta \leq w$ deletions that are localized within a single window of size at most $w$ bits. Block I: The binary message of length $k$ bits is chunked into adjacent blocks of length $\ell$ bits each, and each block is mapped to its corresponding symbol in $G F(q)$ where $q=2^{\ell}$. Block II: The resulting string is encoded using a systematic $(k / \ell+c, k / \ell) q$-ary MDS code where $c$ is the number of parity symbols and $q=k>k / \ell+c$. Block III: The symbols in $G F(q)$ are mapped to their binary representations. Block IV: A buffer of $w$ zeros followed by a single one is inserted between the systematic and the parity bits.
4. Unrestricted deletions: at most $\delta \geq 1$ deletions can occur anywhere in the transmitted string.

We denote by $k$ and $n$ the lengths in bits of the message and codeword, respectively. The regime we study in this chapter is when $w=o(k)$. The encoding block diagram of GC codes for correcting deletions that are localized within a single window $(z=1)$ is shown in Fig. 3.1. We denote binary and $q$-ary vectors by lower and upper case bold letters respectively, and random variables by calligraphic letters. All logarithms in this chapter are of base 2 .

For the aforementioned deletion models, we study the setting where the positions of the deletions are arbitrary but are chosen independently of the codeword. We also assume that the information message is uniform iid. We denote by $F$ the event that corresponds to a decoding failure, i.e., when the decoder cannot make a correct decision and outputs a "failure to decode" error message. $\operatorname{Pr}(F)$ is the probability of decoding failure where this probability is for a uniform iid message and for arbitrary deletion positions that are chosen independently of the codeword.

The notations used in this chapter are summarized in Table I.

| Variable | Description |
| :---: | :--- |
| $\mathbf{u}$ | message |
| $k$ | length of the message in bits |
| $\mathbf{x}$ | codeword |
| $n$ | length of codeword in bits |
| $w$ | size of the deletion window |
| $\delta$ | number of deletions |
| $c$ | number of MDS parity symbols |
| $\ell$ | chunking length used for encoding (Fig. 3.1) |
| $q$ | field size given by $2^{\ell}$ |

Table 3.1: Summary of the notations used in the chapter.

### 3.3 Main Results

In this section, we state our two main results in this chapter and compare them to the most recent works on other deletion models. Theorem 3.1 and Theorem 3.2 cover the case of deletions that are localized in a single window $(z=1)$, with $w=o(\log k)$ and $w=\Omega(\log k)$, respectively. We discuss the generalization of these theorems to $z>1$ windows in Section 3.7.

Theorem 3.1 (Single window with $w=o(\log k)$ ). Guess $\xi^{3}$ Check ( $G C$ ) codes can correct in polynomial time $\delta \leq w$ deletions that are localized within a single window of size at most $w$ bits, where $w=o(\log k)$. The code has the following properties:

1. Redundancy: $n-k=(c+1) \log k+1$ bits,
2. Encoding complexity is $\mathcal{O}(k \log k)$, and decoding complexity is $\mathcal{O}\left(k^{2}\right)$,
3. Probability of decoding failure: $\operatorname{Pr}(F) \leq k^{-(c-4)} / \log k$,
where $c$ is a code parameter that represents the number of parity symbols.

Theorem 3.2 (Single window with $w=\Omega(\log k)$ ). Guess \& Check (GC) codes can correct in polynomial time $\delta \leq w$ deletions that are localized within a single window of size at most $w$ bits, where $w=\Omega(\log k)$ and $w=o(k)$. The code has the following properties:

1. Redundancy: $n-k=(c+1) w+1$ bits,
2. Encoding complexity is $\mathcal{O}(k w)$, and decoding complexity is $\mathcal{O}\left(k^{2}\right)$,
3. Probability of decoding failure: $\operatorname{Pr}(F) \leq(k / w) 2^{-w(c-3)}$,
where $c$ is a code parameter that represents the number of parity symbols.

Corollary 3.3. For both regimes $w=o(\log k)$ and $w=\Omega(\log k)$ described in Theorems 3.1 and 3.2, the probability of decoding failure of GC codes for a single window of localized deletions vanishes asymptotically in $k$ for $c \geq 4$.

The proofs of Theorems 3.1 and 3.2 are provided in Section 3.8. The proof of Corollary 3.3 follows directly from replacing the value of $c$ is the expression of the probability of decoding failure in Theorems 3.1 and 3.2 and is therefore omitted. The code properties in Theorems 3.1 and 3.2 show that:

1. The redundancy is logarithmic in $k$ for $w=o(\log k)$; and linear in $w$ for $w=\Omega(\log k)$. One can easily verify that this corresponds to a code rate $R=k / n$ that is asymptotically optimal for both regimes, i.e., $R \rightarrow 1$ as $k \rightarrow+\infty$.
2. The decoding complexity is quadratic; and the encoding complexity is near-linear for small $w$, and subquadratic for large $w$.
3. The probability of decoding failure vanishes asymptotically in $k$ for $c \geq 4$. Moreover, this probability decreases exponentially in $c$ for a fixed $k$.

In terms of the code construction, the only difference between the two regimes: $w=o(\log k)$ and $w=\Omega(\log k)$, is the choice of the value of the chunking length $\ell$ (Fig. 3.1). For $w=\Omega(\log k)$ we set $\ell=w$ (Theorem 3.2), whereas for $w=o(\log k)$ we set $\ell=\log k$ (Theorem 3.1). The reason we differentiate between these two regimes is because our analysis in Section 3.8 shows that $\ell=\Omega(\log k)$ is always required to guarantee that GC codes have an asymptotically vanishing probability of decoding failure. So although the choice of $\ell=w$ gives a good trade-off between redundancy and decoding complexity as we explain in Section 3.5, it does not guarantee a low probability of error when $w=o(\log k)$.

Remark 3.4. The bounds on the probability of decoding failure in Theorems 3.1 and 3.2 hold for any window position, and any $\delta \leq w$ deletion positions within this window, that are chosen independently of the codeword. Hence, the same result can be also obtained for any random distribution on the positions of the window and the deletions (like the uniform distribution for example), by applying the law of total probability.

Comparison to recent work: As previously mentioned, there has been many works in the literature on other deletion models such as the unrestricted and the bursty models. Next, we compare our results on localized deletions (Theorems 3.1 and 3.2) to the most recent works on these models to provide helpful context to the reader.

Recently, Sima and Bruck [72] introduced optimal codes that can correct $\delta$ unrestricted deletions with zero-error, where $\delta$ is a constant that is fixed with respect to the blocklength. The redundancy of these codes is $\mathcal{O}(\delta \log n)$, which is optimal when compared to Levenshtein's bound. The encoding/decoding complexity is $\mathcal{O}\left(n^{2 \delta+1}\right)$, which is polynomial in $n$ and exponential in $\delta$.

The work that is closest to our work in spirit is the one by Schoeny et al. [43] where the authors introduce codes that can correct a burst of $\delta$ deletions with zero-error, where $\delta$ is a constant that is fixed with respect to the blocklength. For the case of bursts of fixed lengths (exactly $\delta$ deletions), the authors prove the existence of codes that have at most $\log n+(\delta-1) \log (\log n)+$ $\delta-1$ redundancy, for sufficiently large $n$. As for the case of bursts of variable lengths (at most $\delta$ deletions), the redundancy is at most $(\delta-1) \log n+\left(\binom{\delta}{2}-1\right) \log (\log n)+\binom{\delta}{2}+\log (\log \delta)$. The authors in [43] do not provide an explicit polynomial time encoding algorithm for these codes.

Comparing our work on localized deletions (Theorems 3.1 and 3.2) to the works mentioned above, we can observe the following.

1. The number of deletions in both [72] and [43] is constant, whereas in our work the number of localized deletions $w=o(k)$ grows with the message length.
2. In the regime where the number of the deletions is constant, the redundancy in our
construction is lower than the redundancy in [72], which is intuitive since the deletions in the localized model are restricted. Furthermore, in the same regime, the redundancy of our codes is lower than the redundancy in [43] for variable bursts, and can also be lower than the redundancy for fixed bursts for certain values of $\delta$ and $n$.
3. Our codes are explicit, where the encoding complexity is near-linear and the decoding complexity is quadratic. These complexities are much lower than that of the codes in [72]. Furthermore, as previously mentioned, the codes in [43] do not have an explicit encoding algorithm.
4. The gains provided by our codes come at the expense of a small probability of decoding failure that vanishes asymptotically. The codes in [72] and [43] have zero-error.

### 3.4 Encoding and Decoding Examples

In this section, we provide encoding and decoding examples of GC codes for correcting $\delta \leq w$ deletions that are localized within a single window of size $w=\log k$ bits. The chunking length (Fig. 3.1) is set to $\ell=w=\log k$.

Example 3.2 (Encoding). Consider a message $\mathbf{u}$ of length $k=16$ given by $\mathbf{u}=1100101001111000$. $\mathbf{u}$ is encoded by following the different encoding blocks illustrated in Fig. 3.1.

1) Binary to $q$-ary (Block I, Fig. 3.1). The message $\mathbf{u}$ is chunked into adjacent blocks of length $\ell=\log k=4$ bits each,

$$
\mathbf{u}=\underbrace{\underbrace{\text { block 1 }} 1100}_{\alpha^{6}} \underbrace{\underbrace{\text { block 2 }} 0}_{\alpha^{9}} \underbrace{\underbrace{\text { block 3 }}}_{\alpha^{10}} \underbrace{\underbrace{\text { block } 4}}_{\alpha^{3}} \underbrace{1000} .
$$

Each block is then mapped to its corresponding symbol in $G F(q), q=2^{\ell}=2^{4}=16$. This results in a string $\mathbf{U}$ which consists of $k / \log k=4$ symbols in $G F(16)$. The extension field used here has a primitive element $\alpha$, with $\alpha^{4}=\alpha+1$. Hence, we obtain $\mathbf{U}=\left(\alpha^{6}, \alpha^{9}, \alpha^{10}, \alpha^{3}\right) \in G F(16)^{4}$. 2) Systematic MDS code (Block II, Fig. 3.1). U is then encoded using a systematic ( $k / \log k+$ $c, k / \log k)=(7,4)$ MDS code over $G F(16)$, with $c=3$. The encoded string is denoted by
$\mathbf{X} \in G F(16)^{3}$ and is given by multiplying $\mathbf{U}$ by the following code generator matrix ${ }^{3}$

$$
\mathbf{X}=\left(\alpha^{6}, \alpha^{9}, \alpha^{10}, \alpha^{3}\right)\left(\begin{array}{ccccccc}
1 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & \alpha & \alpha^{2} \\
0 & 0 & 1 & 0 & 1 & \alpha^{2} & \alpha^{4} \\
0 & 0 & 0 & 1 & 1 & \alpha^{3} & \alpha^{6}
\end{array}\right)=\left(\alpha^{6}, \alpha^{9}, \alpha^{10}, \alpha^{3}, \alpha^{14}, \alpha^{3}, \alpha\right)
$$

3) $Q$-ary to binary (Block III, Fig. 3.1). The binary representation of $\mathbf{X}$, of length $n=$ $k+3 \log k=28$ bits, is 1100101001111000100110000001.
4) Adding a buffer of $w+1$ bits (Block IV, Fig. 3.1). A buffer of $w=\log k=4$ zeros followed by single one is inserted between the systematic and parity bits. The binary codeword to be transmitted is of length 33 bits and is given by

$$
\mathbf{x}=1100101001111000 \stackrel{\text { buffer }}{\stackrel{\text { b0001 }}{ }} 100110000001
$$

Now we explain the high-level idea of decoding. Since $\delta \leq w$ and $w=\ell=\log k$, then the $\delta$ deletions can affect at most two adjacent blocks in $\mathbf{x}$. The goal of the decoder is to recover the systematic part of $\mathbf{x}$, so it considers following guesses: 1) blocks 1 and 2 are affected by the deletions; 2) blocks 2 and 3 are affected; 3) blocks 3 and 4 are affected. For each of these guesses the decoder: (i) Chunks the received sequence based on its guess on the locations of the two affected blocks. (ii) Considers the two affected blocks erased and decodes them by using the first two MDS parity symbols. (iii) Checks whether the decoded string is consistent with the third MDS parity and with the received sequence.

Example 3.3 (Decoding). Suppose that the $7^{\text {th }}, 9^{\text {th }}$ and $10^{\text {th }}$ bit of $\mathbf{x}$ are deleted. Hence, the decoder receives the following 30 bit string $\mathbf{y}$,

$$
\mathbf{y}=110010011100000001100110000001
$$

[^15]Note that the window size $w$ is known at the decoder, and the number of deletions $\delta$ can be determined by the difference between $n$ (code parameter) and the length of the received string y. In this example, we have $w=\log k=4$ and $\delta=3$. Moreover, the localized $\delta \leq w$ deletions cannot affect both the systematic and the parity bits simultaneously, since these two are separated by a buffer of size $w+1$ bits. Therefore, we consider the following two scenarios: (i) If the deletions affected the parity bits, then the decoder simply outputs the systematic bits ${ }^{4}$. (ii) If the deletions affected the systematic bits, then the decoder goes over the guesses as explained previously.

The buffer is what allows the decoder to determine which of the two previous scenarios to consider. The decoder observes the $(k+w-\delta+1)^{t h}=18^{\text {th }}$ bit in $\mathbf{y}$,

## 110010011100000001100110000001.

Based on the value of the observed bit, the decoder can determine whether the deletions affected the systematic bits or not. The previous operation can be done with zero-error, we explain it in more detail in Section 3.5. In this example, the fact that the observed bit is a one indicates that the one in the buffer has shifted $\delta$ positions to the left. Hence, the decoder considers that the deletions have affected the systematic bits, and thus proceeds with making its guesses as we explain next. Henceforth, the buffer is removed from the string.

The decoder goes through all the possible $k / \log k-1=3$ cases (guesses), where in each case $i, i=1, \ldots, 3$, the deletions are assumed to have affected blocks $i$ and $i+1$, and $\mathbf{y}$ is chunked accordingly. Given this assumption, symbols $i$ and $i+1$ are considered erased and erasure decoding is applied over $G F(16)$ to recover these two symbols. Without loss of generality, we assume that the first two parities $p_{1}=\alpha^{14}$ and $p_{2}=\alpha^{3}$ are used for decoding the two erasures. The decoded $q$-ary string in case $i$ is denoted by $\mathbf{Y}_{\mathbf{i}} \in G F(16)^{4}$, and its binary representation is denoted by $\mathbf{y}_{\mathbf{i}} \in G F(2)^{16}$. The three cases are shown below:

[^16]Case 1: The deletions are assumed to have affected blocks 1 and 2. Hence, $\mathbf{y}$ is chunked as follows

$$
\underbrace{11001}_{\mathcal{E}} \underbrace{0011}_{\alpha^{4}} \underbrace{1000}_{\alpha^{3}} \underbrace{1001}_{\alpha^{14}} \underbrace{1000}_{\alpha^{3}} \underbrace{0001}_{1},
$$

where $\mathcal{E}$ denotes the bits corresponding to symbols 1 and 2 that are considered to be erased. Applying erasure decoding over $G F(16)$, the recovered values of symbols 1 and 2 are $\alpha^{2}$ and $\alpha^{5}$, respectively. Hence, the decoded $q$-ary string $\mathbf{Y}_{\mathbf{1}} \in G F(16)^{4}$ is

$$
\mathbf{Y}_{\mathbf{1}}=\left(\alpha^{2}, \alpha^{5}, \alpha^{4}, \alpha^{3}\right)
$$

Its equivalent in binary $\mathbf{y}_{\mathbf{1}} \in G F(2)^{16}$ is

$$
\mathbf{y}_{\mathbf{1}}=\underbrace{0100}_{\alpha^{2}} \underbrace{0110}_{\alpha^{5}} \underbrace{0011}_{\alpha^{4}} \underbrace{1000}_{\alpha^{3}} .
$$

Notice that the concatenated binary representation of the two decoded erasures (01000110), is not a supersequence of the sub-block (11001), which was denoted by $\mathcal{E}$. Hence, the decoder can immediately point out that the assumption in this case is wrong, i.e., the deletions did not affect blocks 1 and 2. Throughout the chapter we refer to such cases as impossible cases. Another way for the decoder to check whether this case is possible is to test if $\mathbf{Y}_{\mathbf{1}}$ is consistent with the third parity $p_{3}=1$. However, the computed parity is

$$
\left(\alpha^{2}, \alpha^{5}, \alpha^{4}, \alpha^{3}\right)\left(1, \alpha^{2}, \alpha^{4}, \alpha^{6}\right)^{T}=0 \neq 1
$$

Therefore, this is an additional reason which shows that case 1 is impossible.
Case 2: The deletions are assumed to have affected blocks 2 and 3, so the sequence is chunked as follows

$$
\underbrace{1100}_{\alpha^{6}} \underbrace{10011}_{\mathcal{E}} \underbrace{1000}_{\alpha^{3}} \underbrace{1001}_{\alpha^{14}} \underbrace{1000}_{\alpha^{3}} \underbrace{0001}_{1} .
$$

Applying erasure decoding, the recovered values of symbols 2 and 3 are $\alpha^{9}$ and $\alpha^{10}$, respectively.

The decoded binary string is

$$
\mathbf{y}_{\mathbf{2}}=\underbrace{1100}_{\alpha^{6}} \underbrace{1010}_{\alpha^{9}} \underbrace{0111}_{\alpha^{10}} \underbrace{1000}_{\alpha^{3}} .
$$

In this case, the concatenated binary representation of the two decoded erasures (10100111) is a supersequence of the sub-block (10011). Moreover, it is easy to verify that the decoded string is consistent with the third parity $p_{3}=1$. Therefore, we say that case 2 is possible.

Case 3: The deletions are assumed to have affected blocks 3 and 4, so the sequence is chunked as follows

$$
\underbrace{1100}_{\alpha^{6}} \underbrace{1001}_{\alpha^{14}} \underbrace{110000}_{\mathcal{E}} \underbrace{1001}_{\alpha^{14}} \underbrace{1000}_{\alpha^{3}} \underbrace{0001}_{1} .
$$

The decoded binary string is

$$
\mathbf{y}_{\mathbf{3}}=\underbrace{1100}_{\alpha^{6}} \underbrace{1001}_{\alpha^{14}} \underbrace{1100}_{\alpha^{6}} \underbrace{0000}_{0} .
$$

In this case, the concatenated binary representation of the two decoded erasures (11000000) is a supersequence of the sub-block (11000). However, it is easy to verify that the decoded string is not consistent with $p_{3}=1$. Therefore, case 3 is impossible.

After going through all the cases, case 2 stands alone as the only possible case. So the decoder declares successful decoding and outputs $\mathbf{y}_{\mathbf{2}}\left(\mathbf{y}_{\mathbf{2}}=\mathbf{u}\right)$.

Remark 3.5. Sometimes the decoder may find more than one possible case resulting in different decoded strings. In that situation, the decoder cannot know which of the cases is the correct one, so it declares a decoding failure. Although a decoding failure may occur, Theorems 3.1 and 3.2 indicate that its probability vanishes as length of the message $k$ goes to infinity.

### 3.5 Encoding and Decoding for Localized Deletions

As previously mentioned, our schemes for correcting localized deletions extend from the encoding and decoding schemes of Guess \& Check (GC) codes discussed in Chapter 2, which are designed
for correcting $\delta$ deletions that are not necessarily localized (i.e., unrestricted). In this section we explain how we exploit the localized nature of the deletions to modify these schemes and obtain codes having the properties shown in Section 3.3. Throughout our discussion, we discuss the similarities and the differences with respect to our construction presented in Chapter 2, Theorem 2.7.

Next, we explain the encoding and decoding using GC codes for the following two models:
(1) Unrestricted deletions; and (2) Localized deletions in a single window ( $z=1$ ).

### 3.5.1 Encoding using GC codes

The first three encoding blocks are the same as the ones shown in Fig. 3.1 for both models mentioned above. For unrestricted deletions, the choice of the chunking length $\ell$ presents a trade-off between redundancy, complexity, and probability of decoding failure. For localized deletions, we specify the value of $\ell$ based on the size of the window $w$. Namely,

$$
\ell= \begin{cases}\log k & \text { if } w=o(\log k)  \tag{3.1}\\ w & \text { if } w=\Omega(\log k) \text { and } w=o(k)\end{cases}
$$

As for the last encoding block (Block IV, Fig. 3.1):

## Unrestricted deletions

The parity bits are encoded using a $(\delta+1)$ repetition code, where each parity bit is repeated $(\delta+1)$ times. The repetition code protects the parities against any deletions, and allows them to be recovered at the decoder.

## Localized deletions in a single window $(z=1)$

The systematic and parity bits are separated by a buffer of size $w+1$ bits, which consists of $w$ zeros followed by a single one. In the upcoming decoding section, we explain how the decoder uses this buffer to detect whether the deletions affected the systematic bits or not.

### 3.5.2 Decoding using GC codes

## Unrestricted deletions

The approach presented in [71] for decoding up to $\delta$ unrestricted deletions is the following:
(a) Decoding the parity bits: the decoder recovers the parity bits which are protected by a $(\delta+1)$ repetition code.
(b) The guessing part: the number of possible ways to distribute the $\delta$ deletions among the $k / \ell$ blocks is $t=\binom{k / \ell+\delta-1}{\delta}$. These possibilities are indexed by $i, i=1, \ldots, t$, and each possibility is referred to by case $i$.
(c) The checking part: for each case $i, i=1, \ldots, t$, the decoder: (i) Chunks the sequence based on the corresponding assumption on the locations of the $\delta$ deletions. (ii) Considers the affected blocks erased and maps the remaining blocks to their corresponding symbols in $G F(q)$.
(iii) Decodes the erasures using the first $\delta$ parity symbols. (iv) Checks whether the case is possible or not by testing if the decoded string is consistent with the received string and with the last $c-\delta$ parity symbols. The criteria used to check if a case is possible or not are given in Definition 3.6.

Definition 3.6. For $\delta$ deletions, a case $i, i=1, \ldots, t$, is said to be possible if it satisfies the following two criteria simultaneously. Criterion 1: the decoded $q$-ary string in case $i$ is consistent with the last $c-\delta$ parities simultaneously. Criterion 2: the binary representations of all the decoded erasures are supersequences of their corresponding sub-blocks.

## Localized deletions in a single window $(z=1)$

Consider the case of decoding $\delta \leq w$ deletions that are localized within a single window of size at most $w$ bits. Since $w \leq \ell$ from (3.1), then the deletions can affect at most two adjacent blocks in the codeword. Therefore, in terms of the GC code construction, correcting the $\delta \leq w$ deletions corresponds to decoding at most two block erasures. Hence, the localized nature of the deletions enables the following simplifications to the scheme: (i) The total number of cases to be checked by the decoder is reduced to $k / \ell-1$ since at most two adjacent blocks (out of $k / \ell$
blocks) can be affected by the deletions. (ii) Instead of protecting the parity bits by a ( $\delta+1$ ) repetition code, it is sufficient to separate them from the systematic bits by inserting a buffer of size $w+1$ bits, composed of $w$ zeros followed by a single one.

Now we explain the decoding steps. Note that the $\delta \leq w$ localized deletions cannot affect the systematic and the parity bits simultaneously since these two are separated by a buffer of size $w+1$ bits. The decoder uses this buffer to detect whether the deletions have affected the systematic bits or not. The buffer is composed of $w$ zeros followed by a single one, and its position ranges from the $(k+1)^{t h}$ bit to the $(k+w+1)^{t h}$ bit of the transmitted string. Let $\mathbf{y}_{\lambda}$ be the bit in position $\lambda$ in the received string, where $\lambda \triangleq k+w-\delta+1$. The decoder observes $\mathbf{y}_{\lambda}{ }^{5}:(1)$ If $\mathbf{y}_{\lambda}=1$, then this means that the one in the buffer has shifted $\delta$ positions to the left because of the deletions, i.e., all the deletions occurred to the left of the one in the buffer. In this case, the decoder considers that the deletions affected the systematic bits, and therefore proceeds to the guessing and checking part. It applies the same steps as in the case of $\delta$ deletions, while considering a total of $k / \ell-1$ cases, each corresponding to two adjacent block erasures. In each case, the last $c \ell$ bits of the received string (parities) are used to decode the first $k-\delta$ bits. (2) If $\mathbf{y}_{\lambda}=0$, then this indicates that the $\delta$ deletions occurred to the right of the first zero in the buffer, i.e., the systematic bits were unaffected. In this case, the decoder simply outputs the first $k$ bits of the received string.

### 3.5.3 Discussion

## Choice of the chunking length $\ell$ for localized deletions

Recall that the choice of the value of $\ell$ affects the redundancy of the code which is given by $c \ell+w+1$ (Fig.3.1). For this reason, we would like keep $\ell$ as small as possible in order to minimize the redundancy of the code. However, small values of $\ell$ can be problematic due to the following reasons: (i) If $\ell$ is small, then the number of blocks given by $k / \ell$ is large. As a result, the number of cases to be checked by the decoder is large, and therefore this leads

[^17]to an increase in the decoding complexity. (ii) Our theoretical analysis in Section 3.8 shows that $\ell=\Omega(\log k)$ is required in order to guarantee an asymptotically vanishing probability of decoding failure (Remark 3.13). Hence, small values of $\ell$ do not guarantee a high probability of successful decoding. Furthermore, the theoretical analysis also shows that larger values of $\ell$ lead to a better upper bound on the probability of failure.

Due to the reasons mentioned above, we specify $\ell$ based on (3.1). For $w=\Omega(\log k)$, we adopt the choice of $\ell=w$ since it gives a good trade-off between the code properties and simplifies the theoretical analysis. As for $w=o(\log k)$, we set $\ell=\log k$ since the choice of $\ell=w$ in this regime does not guarantee an asymptotically vanishing probability of decoding failure.

Remark 3.7. Since the redundancy is linear in $\ell$, then the condition of $\ell=\Omega(\log k)$ implies that the number of redundant bits should be at least logarithmic in order to correct the deletions successfully with high probability. This is in fact intuitive, because even for the case of one deletion, one would at least need to communicate the position of the deletion within the string, which would require a logarithmic number of redundant bits.

## The minimum required number of MDS parities $c$

The decoding scheme for localized deletions requires $c \geq 3$ MDS parity symbols (2 for decoding the hypothetical block erasures and at least 1 for checking). However, the theoretical results in Theorems 3.1 and 3.2 suggest that $c \geq 4$ parity symbols are needed in order to guarantee an asymptotically vanishing probability of decoding failure. This discrepancy is due to the looseness of the theoretical upper bound on the probability of decoding failure for $c=3$. We discuss this in detail in Section 3.6, where we analyze the performance of GC codes for $c=3$ using numerical simulations.

## Comparison to the case of unrestricted deletions

The encoding and decoding schemes for localized deletions result in the code properties shown in Theorems 3.1 and 3.2. Compared to the case of unrestricted deletions (Theorem 2.7), we observe the following: (i) The construction presented for localized deletions enables correcting a number of deletions that grows with the message length $k$, as opposed to a constant number
of unrestricted deletions. (ii) In the case of localized deletions, the parity bits can be recovered with zero-error without the use of a repetition code, which enhances the redundancy. (iii) The decoding complexity is quadratic for the case of localized deletions, as opposed to polynomial of degree $\delta$ for unrestricted deletions.

### 3.6 Simulation Results

We tested the performance of GC codes for correcting deletions that are localized within a single window of size $w=\log k$ bits. We performed numerical simulations for messages of length $k=128,256,512,1024,2048$, and 4096 bits. We also compared the resulting empirical probability of decoding failure to the upper bound in Theorem 3.2.

### 3.6.1 Results for $c=3$ MDS parity symbols

Recall that the GC encoder (Fig. 3.1) adds $c>2$ MDS parity symbols resulting in $(c+1) \log k+1$ redundant bits when $\ell=w=\log k$. Here, we show the simulation results for the case of $c=3$ (Fig. 3.2), which corresponds to the minimum redundancy of GC codes.

| Config. | $c=3$ |  |  |  |
| :---: | :---: | :---: | :---: | :---: |
| $k$ | $R$ | $\operatorname{Pr}(F)$ |  |  |
|  |  | $\delta=0.5 w$ | $\delta=0.75 w$ | $\delta=w$ |
| 128 | 0.82 | $9.06 e^{-3}$ | $3.19 e^{-2}$ | $4.19 e^{-2}$ |
| 256 | 0.89 | $5.06 e^{-3}$ | $2.43 e^{-2}$ | $4.11 e^{-2}$ |
| 512 | 0.93 | $3.81 e^{-3}$ | $2.36 e^{-2}$ | $3.96 e^{-2}$ |
| 1024 | 0.96 | $2.35 e^{-3}$ | $2.34 e^{-2}$ | $3.75 e^{-2}$ |
| 2048 | 0.98 | $2.09 e^{-3}$ | $2.28 e^{-2}$ | $3.59 e^{-2}$ |
| 4096 | 0.99 | $9.7 e^{-4}$ | $1.31 e^{-2}$ | $3.36 e^{-2}$ |

Figure 3.2: The figures show the code rate $R=k / n$ and the empirical probability of decoding failure $\operatorname{Pr}(F)$ of GC codes for different message lengths $k$ and number of deletions $\delta$. The $\delta$ deletions are localized within a window of size $w=\log k$ bits. The values of $R$ are rounded to the nearest $10^{-2}$. The results of $\operatorname{Pr}(F)$ are averaged over $10^{5}$ runs of simulations. In each run, a message $\mathbf{u}$ chosen uniformly at random is encoded into the codeword $\mathbf{x}$. The positions of the window and the deletions in $\mathbf{x}$ are correspondingly chosen uniformly at random. The resulting string is then decoded.

### 3.6.2 Results for $c=4$ and $c=5$ MDS parity symbols

The table in Fig. 3.3 shows the simulation results for $c=4$ with the same experimental setup described in Fig. 3.2. We also simulated the case of $c=5$ for the same values of $k$ and we were not able to detect any failure within $10^{5}$ runs of simulations.

| Config. | $c=4$ |  |  |  |  |
| :---: | :---: | :---: | :---: | :---: | :---: |
| $k$ | $R$ | $\operatorname{Pr}(F)$ |  |  |  |
|  |  | $\delta=0.5 w$ | $\delta=0.75 w$ | $\delta=w$ | Bound |
| 128 | 0.78 | $7.0 e^{-5}$ | $2.3 e^{-4}$ | $2.7 e^{-4}$ | $1.4 e^{-1}$ |
| 256 | 0.86 | $3.0 e^{-5}$ | $6.0 e^{-5}$ | $1.3 e^{-4}$ | $1.3 e^{-1}$ |
| 512 | 0.92 | $1.0 e^{-5}$ | $3.0 e^{-5}$ | $8.0 e^{-5}$ | $1.1 e^{-1}$ |
| 1024 | 0.95 | $1.0 e^{-5}$ | $3.0 e^{-5}$ | $5.0 e^{-5}$ | $1.0 e^{-1}$ |
| 2048 | 0.97 | 0 | 0 | $3.0 e^{-5}$ | $9.1 e^{-2}$ |
| 4096 | 0.99 | 0 | 0 | 0 | $8.3 e^{-2}$ |

Figure 3.3: The values of the code rate $R$ and the empirical probability of failure $\operatorname{Pr}(F)$ for different message lengths $k$ and number of localized deletions $\delta$, for the same experimental setup described in Fig. 3.2. The table also shows that GC codes perform better than what the theoretical upper bound in Theorem 3.2 indicates.

### 3.6.3 Comparison to the theoretical upper bound in Theorem 3.2

To guarantee an asymptotically vanishing probability of decoding failure, the upper bound in Theorem 3.2 requires that $c \geq 4$. In fact, the bound shows that the probability of decoding failure decreases logarithmically in $k$ for $c=4$, whereas for $c>4$, the probability decreases polynomially. Therefore, we make a distinction between the following three regimes. (i) $c=3$ : Here, the theoretical upper bound is trivial, whereas in the simulation results in Fig. 3.2 we observe that the probability of decoding failure is at most of the order of $10^{-2}$. (ii) $c=4$ (Fig. 3.3): The upper bound ranges from the order of $10^{-1}$ for $k=128$ to the order of $10^{-2}$ for $k=4096$, whereas the probability of decoding failure recorded in the simulations is at most of the order of $10^{-4}$ for $k=128$. Moreover, for $k=4096$, no failures were detected within $10^{5}$ runs of simulations. (iii) $c>4$ : For $k=4096$, the upper bound is of the order of $10^{-5}$ for $c=5$, and of the order of $10^{-9}$ for $c=6$. In the simulations, no decoding failure was detected within $10^{5}$
runs for $c=5$. In general, the simulations show that GC codes perform better than what the upper bound indicates. The looseness of the bound is due to the fact that the effect of Criterion 2 (Definition 3.6) is neglected while deriving the bound (Section 3.8.3). Furthermore, notice that for a fixed $k$, the empirical $\operatorname{Pr}(F)$ increases as the number of deletions within the window increases. This dependence on $\delta$ is not reflected in the theoretical bound, since this bound is derived for the worst case of $\delta=w$, as we discuss in Section 3.8.3.

### 3.7 Correcting Deletions Localized in Multiple Windows

In this section, we discuss how we generalize the previous results to the case where the deletions are localized in $z>1$ windows. In terms of the encoding scheme, the first three encoding blocks remain the same as the ones for shown in Fig. 3.1, where the chunking length $\ell$ is specified based on (3.1). As for the last encoding block (Block IV, Fig. 3.1), we use a $(z w+1)$ repetition code to encode the parity bits. For decoding, the decoder first recovers the parity bits which are protected by the repetition code and then applies the guess and check method.

The starting locations of the $z$ windows are distributed among the $k / \ell$ systematic blocks of the codeword. Furthermore, there are up to $z w$ bit deletions that are distributed among these $z$ windows. Therefore, the total number of cases to be checked by the decoder is

$$
t=\mathcal{O}\left(\binom{k / \ell}{z}\binom{z w+z-1}{z}\right)
$$

Recall that $z>1$ is a constant and $w \leq \ell$. Therefore,

$$
\begin{equation*}
t=\mathcal{O}\left(\frac{k^{z}}{\ell^{z}} \cdot(z w)^{z}\right)=\mathcal{O}\left(k^{z}\right) \tag{3.2}
\end{equation*}
$$

Note that the exact value of $t$ depends on the sizes of the deletion windows. Nevertheless, as shown above, the order of $t$ is polynomial in $k$. The decoder goes over all these cases and applies the guess \& check method explained previously. The resulting code properties are given
in Theorems 3.8 and 3.9 for $w=o(\log k)$ and $w=\Omega(\log k)$, respectively.

Theorem 3.8 (Multiple windows with $w=o(\log k)$ ). Guess $\S$ Check ( $G C$ ) codes can correct in polynomial time deletions that are localized within a constant number of windows $z>1$, where the size of each window is at most $w=o(\log k)$ bits, and the number of deletions in each window is at most $\delta \leq w$. The code has the following properties:

1. Redundancy: $n-k=c(z w+1) \log k$ bits,
2. Encoding complexity is $\mathcal{O}(k \log k)$, and decoding complexity is $\mathcal{O}\left(k^{z+1} \log k\right)$,
3. Probability of decoding failure: $\operatorname{Pr}(F)=\mathcal{O}\left(k^{-(c-4 z)}\right)$,
where $c$ is a code parameter that represents the number of parity symbols.

Theorem 3.9 (Multiple windows with $w=\Omega(\log k))$. Guess $\xi^{6}$ Check (GC) codes can correct in polynomial time deletions that are localized within a constant number of windows $z>1$, where the size of each window is at most $w=\Omega(\log k)$ bits, and the number of deletions in each window is at most $\delta \leq w$. The code has the following properties:

1. Redundancy: $n-k=c(z w+1) w$ bits,
2. Encoding complexity is $\mathcal{O}(k w)$, and decoding complexity is $\mathcal{O}\left(k^{z+1} w\right)$,
3. Probability of decoding failure:

$$
\operatorname{Pr}(F)=\mathcal{O}\left(k^{z} 2^{-w(c-3 z)}\right)
$$

where $c$ is a code parameter that represents the number of parity symbols.

Corollary 3.10. For both regimes $w=o(\log k)$ and $w=\Omega(\log k)$ described in Theorems 3.8 and 3.9, the probability of decoding failure of $G C$ codes for $z>1$ windows of localized deletions vanishes asymptotically in $k$ for $c \geq 4 z$.

The proofs of Theorems 3.8 and 3.9 are provided in Appendix B. The proof of Corollary 3.10 follows directly from replacing the value of $c$ is the expression of the probability of decoding failure in Theorems 3.8 and 3.9 and is therefore omitted.

Remark 3.11. The results in Theorems 3.8 and 3.9 are for a fixed number of windows $z>1$, where each window has a size at most $w$ that can grow with the message length $k$. The decoding complexity in this case is polynomial of degree $z+1$. Furthermore, the probability of decoding failure vanishes asymptotically if $c \geq 4 z$.

### 3.8 Proof of Theorem 3.1 and Theorem 3.2

Recall that the chunking length $\ell$ (Fig. 3.1) is specified based on the size of the window $w$, as given in (3.1). In this section, we provide the proofs for the redundancy, complexity, and probability of decoding failure of GC codes in terms of the parameter $\ell$. Substituting $\ell=\log k$ and $\ell=w$ gives the results in Theorems 3.1 and 3.2 , respectively. The proofs provided in this section follow similar techniques as the ones used to prove the main result in Chapter 2 (Theorem 2.7). The analysis requires some modifications to deal with the case of localized deletions. For the sake of completeness, we go over all the steps in detail.

### 3.8.1 Redundancy

The redundancy follows from the construction for the case of deletions that are localized within a single window (Fig. 3.1). The number of redundant bits is $c \ell+w+1$ (parities + buffer).

### 3.8.2 Complexity

The encoding complexity is dominated by the complexity of computing the $c$ MDS parity symbols. Computing one parity symbol involves $k / \ell$ multiplications of symbols in $G F\left(2^{\ell}\right)$, and hence its complexity is $\mathcal{O}\left((k / \ell) \log ^{2}\left(2^{\ell}\right)\right)=\mathcal{O}(k \ell)$. Since $c$ is a constant, the overall encoding complexity is $\mathcal{O}(k \ell)$. The dominant factor in the decoding complexity is the part where the
decoder goes over all the possible cases and applies erasure decoding for each case. Hence, the order of decoding complexity is given by the total number of cases multiplied by the complexity of erasure decoding. Recall that in each case only two erasures within the systematic symbols are to be decoded. These two erasures can be decoded by: (i) Multiplying the unerased systematic symbols by the corresponding encoding vectors and subtracting the obtained results from the corresponding parity symbols. (ii) Inverting a $2 \times 2$ matrix. Therefore, the overall decoding complexity is

$$
\mathcal{O}\left(\left(\frac{k}{\ell}-1\right) \cdot k \ell\right)=\mathcal{O}\left(k^{2}\right) .
$$

### 3.8.3 Probability of Decoding Failure

The probability of decoding failure is computed over all possible $k$-bit messages. Recall that the message $\mathbf{u}$ is uniform iid, i.e., the bits of $\mathbf{u}$ are iid $\operatorname{Bernoulli}(1 / 2)$. The message $\mathbf{u}$ is encoded as shown in Fig. 3.1. Consider $\delta \leq w$ deletions that are localized within a window of size at most $w$ bits. Due to the chunking length specified in (3.1), the $\delta$ deletions can affect at most two adjacent blocks in the codeword. Therefore, the decoder goes through $k / \ell-1$ cases, where in each case it assumes that certain two adjacent blocks were affected by the deletions. Let $\mathcal{Y}_{i}$ be the random variable representing the $q$-ary string decoded in case $i, i=1,2, \ldots, k / \ell-1$. Let $\mathbf{Y} \in G F(q)^{k / \ell}$ be a realization of the random variable $\mathcal{Y}_{i}$. We denote by $\mathcal{P}_{r} \in G F(q), r=1,2, \ldots, c$, the random variable representing the $r^{t h}$ MDS parity symbol, and let $p_{r}$ be a realization of the random variable $\mathcal{P}_{r}$. Also, let $\mathbf{G}_{\mathbf{r}} \in G F(q)^{k / \ell}$ be the MDS encoding vector responsible for generating $\mathcal{P}_{r}$. For $r=1, \ldots, c$, we define the following random sets

$$
\begin{aligned}
& \mathrm{A}_{\mathrm{r}} \triangleq\left\{\mathbf{Y} \in G F(q)^{k / \ell} \mid \mathbf{G}_{\mathbf{r}}^{\mathbf{T}} \mathbf{Y}=\mathcal{P}_{r}\right\}, \\
& \mathrm{A}_{1}^{\mathrm{c}} \triangleq \mathrm{~A}_{1} \cap \mathrm{~A}_{2} \cap \ldots \cap \mathrm{~A}_{\mathrm{c}} .
\end{aligned}
$$

For any set of given parities $p_{1}, \ldots, p_{c}, \mathrm{~A}_{\mathrm{r}}$ and $\mathrm{A}_{1}^{\mathrm{c}}$ are affine subspaces of dimensions $k / \ell-1$ and $k / \ell-c$, respectively. Since $q=2^{\ell}$, we have

$$
\begin{equation*}
\left|\mathrm{A}_{\mathrm{r}}\right|=2^{k-\ell} \text { and }\left|\mathrm{A}_{1}^{\mathrm{c}}\right|=2^{k-c \ell} \tag{3.3}
\end{equation*}
$$

$\mathcal{Y}_{i}$ is obtained by decoding two erasures based on the first two parities, therefore, $\mathcal{Y}_{i} \in \mathrm{~A}_{1} \cap \mathrm{~A}_{2}$. Note that $\mathcal{Y}_{i}$ is not necessarily uniformly distributed over $A_{1} \cap A_{2}$. The next claim gives an upper bound on the probability mass function of $\mathcal{Y}_{\boldsymbol{i}}$ for given arbitrary parities.

Claim 3.12. For any case $i, i=1,2, \ldots, k / \ell-1$,

$$
\operatorname{Pr}\left(\mathcal{Y}_{i}=\mathbf{Y} \mid \mathcal{P}_{1}=p_{1}, \mathcal{P}_{2}=p_{2}\right) \leq \frac{1}{2^{k-3 \ell}}
$$

We assume Claim 2.5 is true for now and prove it shortly. Next, we use this claim to prove the upper bound on the probability of decoding failure.

In the general decoding scheme, there are two criteria which determine whether a case is possible or not (Definition 3.6). Here, we upper bound $\operatorname{Pr}(F)$ by taking into account Criterion 1 only. Based on Criterion 1, if a case $i$ is possible, then $\mathcal{Y}_{\boldsymbol{i}}$ satisfies all the $c$ MDS parities simultaneously, i.e., $\mathcal{Y}_{\boldsymbol{i}} \in \mathrm{A}_{1}^{\mathrm{c}}$. Without loss of generality, we assume case 1 is the correct case. A decoding failure is declared if there exists a possible case $j, j=2, \ldots, k / \ell-1$, that leads to a decoded string different than that of case 1 . Namely, $\mathcal{Y}_{\boldsymbol{j}} \in \mathrm{A}_{1}^{\mathrm{c}}$ and $\mathcal{Y}_{\boldsymbol{j}} \neq \boldsymbol{Y}_{\mathbf{1}}$. We first compute the conditional probability of failure for given parities $p_{1}, \ldots, p_{c}$. Note that in this case the set
$\mathrm{A}_{1}^{\mathrm{c}}$ is fixed. Let $t \triangleq k / \ell-1$, we have,

$$
\begin{align*}
\operatorname{Pr}\left(F \mid p_{1}, p_{2}, \ldots, p_{c}\right) & \leq \operatorname{Pr}\left(\bigcup_{j=2}^{t}\left\{\mathcal{Y}_{\boldsymbol{j}} \in \mathrm{A}_{1}^{\mathrm{c}}, \mathcal{Y}_{\boldsymbol{j}} \neq \mathcal{Y}_{\mathbf{1}}\right\} \mid p_{1}, p_{2}, \ldots, p_{c}\right)  \tag{3.4}\\
& \leq \sum_{j=2}^{t} \operatorname{Pr}\left(\mathcal{Y}_{\boldsymbol{j}} \in \mathrm{A}_{1}^{\mathrm{c}}, \mathcal{Y}_{\boldsymbol{j}} \neq \mathcal{Y}_{\mathbf{1}} \mid p_{1}, p_{2}, \ldots, p_{c}\right)  \tag{3.5}\\
& \leq \sum_{j=2}^{t} \operatorname{Pr}\left(\mathcal{Y}_{\boldsymbol{j}} \in \mathrm{A}_{1}^{\mathrm{c}} \mid p_{1}, p_{2}, \ldots, p_{c}\right)  \tag{3.6}\\
& =\sum_{j=2}^{t} \sum_{\mathbf{Y} \in \mathrm{A}_{1}^{\mathrm{c}}} \operatorname{Pr}\left(\mathcal{Y}_{\boldsymbol{j}}=\mathbf{Y} \mid p_{1}, p_{2}\right)  \tag{3.7}\\
& \leq \sum_{j=2}^{t} \sum_{\mathbf{Y} \in \mathrm{A}_{1}^{\mathrm{c}}} \frac{1}{2^{k-3 \ell}}  \tag{3.8}\\
& =\sum_{j=2}^{t}\left|\mathrm{~A}_{1}^{\mathrm{c}}\right| \frac{1}{2^{k-3 \ell}}  \tag{3.9}\\
& =\sum_{j=2}^{t} \frac{2^{k-c \ell}}{2^{k-3 \ell}}  \tag{3.10}\\
& =(t-1) 2^{-(c-3) \ell}  \tag{3.11}\\
& =\left(\frac{k}{\ell}-2\right) 2^{-(c-3) \ell}  \tag{3.12}\\
& \leq \frac{k}{\ell} \times 2^{-(c-3) \ell} \tag{3.13}
\end{align*}
$$

(3.5) follows from applying the union bound. (3.6) follows from the fact that the conditional probability $\operatorname{Pr}\left(\mathcal{Y}_{\boldsymbol{j}} \neq \mathcal{Y}_{\mathbf{1}} \mid \mathcal{Y}_{\boldsymbol{j}} \in \mathrm{A}_{1}^{\mathrm{c}}, p_{1}, \ldots, p_{c}\right) \leq 1$. (3.7) follows from the fact that $\mathcal{Y}_{i} \in \mathrm{~A}_{1} \cap \mathrm{~A}_{2}$ and $\mathrm{A}_{1}^{\mathrm{c}} \subset \mathrm{A}_{1} \cap \mathrm{~A}_{2}$. (3.8) follows from Claim 2.5. (3.10) follows from (3.3). (3.12) follows from the fact that $t=k / \ell-1$. Since (3.13) does not depend on the values of the parities, then the same bound is also obtained for $\operatorname{Pr}(F)$.

Remark 3.13. Notice from (3.13) that $\operatorname{Pr}(F)$ goes zero asymptotically in $k$ if $\ell=\Omega(\log k)$. Hence, for a sub-logarithmic window size, i.e., $w=o(\log k)$, setting $\ell=w$ does not guarantee an asymptotically vanishing probability of decoding failure. To this end, we use $\ell=\log k$ for encoding when $w=o(\log k)$.

### 3.8.4 Proof of Claim 3.12

Recall that $\mathcal{Y}_{i} \in \mathrm{~A}_{1} \cap \mathrm{~A}_{2}$ is the random variable representing the output of the decoder in case $i$. Claim 3.12 gives an upper bound on the probability mass function of $\mathcal{Y}_{\boldsymbol{i}}$ for any $i$ and for given arbitrary parities $\left(p_{1}, p_{2}\right)$. To find the bound in Claim 3.12, we focus on an arbitrary case $i(i$ fixed $)$ that assumes that the deletions have affected blocks $i$ and $i+1$. We observe $\mathcal{Y}_{\boldsymbol{i}}$ for all possible input $k$-bit messages, for a fixed deletion window ${ }^{6}$, and given parities $\left(p_{1}, p_{2}\right)$. Hence, the observed case, the deletion window, and parities are fixed, while the input message varies. In this setting, we determine the maximum number of different inputs that can generate the same output. We call this number $\gamma$. Once we obtain $\gamma$ we can write

$$
\begin{equation*}
\operatorname{Pr}\left(\mathcal{Y}_{i}=\mathbf{Y} \mid W, p_{1}, p_{2}\right) \leq \frac{\gamma}{\left|\mathrm{A}_{1} \cap \mathrm{~A}_{2}\right|}=\frac{\gamma}{2^{k-2 \ell}} \tag{3.14}
\end{equation*}
$$

where $W$ is an arbitrary window of size $w$ bits in which the $\delta$ deletions are localized. We will explain our approach for determining $\gamma$ by going through an example for $k=32$ that can be generalized for any $k$. We denote by $b_{o} \in G F(2), o=1,2, \ldots, k$, the bit of the message $\mathbf{u}$ in position $o$.

Example 3.4. Let $k=32$ and $\delta=w=\ell=\log k=5$. Consider the binary message $\mathbf{u}$ given by

$$
\mathbf{u}=b_{1} \quad b_{2} \ldots b_{32}
$$

Its corresponding $q-a r y$ message $\mathbf{U}$ consists of 7 symbols (blocks) of length $\log k=5$ bits each ${ }^{7}$. The message $\mathbf{u}$ is encoded into a codeword $\mathbf{x}$ as shown in Fig. 3.1. We assume that the first parity is the sum of the systematic symbols and the encoding vector for the second parity is $\left(1, \alpha, \alpha^{2}, \alpha^{3}, \alpha^{4}, \alpha^{5}, \alpha^{6}\right)^{8}$. Suppose that the $\delta=w=5$ deleted bits in $\mathbf{x}$ are $b_{3}$ up to $b_{7}$. Recall that since $\ell=w$, the deletions can affect at most two blocks in $\mathbf{x}$. Suppose that the case

[^18]considered by the decoder is the one that assumes that the deletions affected the $3^{\text {rd }}$ and $4^{\text {th }}$ block (wrong case). The decoder chunks the codeword accordingly, and symbols 3 and 4 are considered to be erased. The rest of the $q$-ary symbols are given by
\[

$$
\begin{aligned}
& S_{1}=\alpha^{4} b_{1}+\alpha^{3} b_{2}+\alpha^{2} b_{8}+\alpha b_{9}+b_{10} \\
& S_{2}=\alpha^{4} b_{11}+\alpha^{3} b_{12}+\alpha^{2} b_{13}+\alpha b_{14}+b_{15} \\
& S_{5}=\alpha^{4} b_{21}+\alpha^{3} b_{22}+\alpha^{2} b_{23}+\alpha b_{24}+b_{25} \\
& S_{6}=\alpha^{4} b_{26}+\alpha^{3} b_{27}+\alpha^{2} b_{28}+\alpha b_{29}+b_{30} \\
& S_{7}=\alpha^{4} b_{31}+\alpha^{3} b_{32}
\end{aligned}
$$
\]

Notice that $S_{1}, S_{2}, S_{5}, S_{6}$ and $S_{7}$ are directly determined by the bits of $\mathbf{u}$ which are chunked at their corresponding positions. Hence, in order to obtain the same output, the bits of the inputs corresponding to these symbols cannot be different. For instance, if two messages differ in the first bit, then they will differ in $S_{1}$ when they are decoded, i.e., these two messages cannot generate the same output. Therefore, we refer to the bits corresponding to these symbols by the term "fixed bits". The "free bits", i.e., the bits which can differ in the input, are the $2 w-\delta=5$ bits corresponding to the erasure $b_{16}, b_{17}, b_{18}, b_{19}, b_{20}$, in addition to the $\delta=5$ deleted bits $b_{3}, b_{4}, b_{5}, b_{6}, b_{7}$. The total number of "free bits" is $2 w=10$, so an immediate upper bound on $\gamma$ is $\gamma \leq 2{ }^{10}$. However, these "free bits" are actually constrained by the linear equations which generate the first two parities. By analyzing these constraints, one can obtain a tighter bound on $\gamma$.

The constraints on the "free bits" are given by the following system of two linear equations in $G F(32)$,

$$
\left\{\begin{array}{l}
\alpha^{2} b_{3}+\alpha b_{4}+b_{5}+\alpha^{4} b_{6}+\alpha^{3} b_{7}  \tag{3.15}\\
\quad+\alpha^{4} b_{16}+\alpha^{3} b_{17}+\alpha^{2} b_{18}+\alpha b_{19}+b_{20}=p_{1}^{\prime} \\
\\
\\
\alpha^{2} b_{3}+\alpha b_{4}+b_{5}+\alpha\left(\alpha^{4} b_{6}+\alpha^{3} b_{7}\right) \\
\quad+\alpha^{3}\left(\alpha^{4} b_{16}+\alpha^{3} b_{17}+\alpha^{2} b_{18}+\alpha b_{19}+b_{20}\right)=p_{2}^{\prime}
\end{array}\right.
$$

where $p_{1}^{\prime}, p_{2}^{\prime} \in G F(32)$ are obtained by the difference between the first and the second parity (respectively) and the part corresponding to the "free bits". To upper bound $\gamma$, we upper bound the number of solutions of the system given by (3.15). Equation (3.15) can be written as follows

$$
\left\{\begin{align*}
B_{1}+B_{2}+B_{3} & =p_{1}^{\prime}  \tag{3.16}\\
B_{1}+\alpha B_{2}+\alpha^{3} B_{3} & =p_{2}^{\prime}
\end{align*}\right.
$$

where $B_{1}, B_{2}$ and $B_{3}$ are three symbols in $G F(32)$ given by

$$
\begin{align*}
& B_{1}=\alpha^{2} b_{3}+\alpha b_{4}+b_{5}  \tag{3.17}\\
& B_{2}=\alpha^{4} b_{6}+\alpha^{3} b_{7}  \tag{3.18}\\
& B_{3}=\alpha^{4} b_{16}+\alpha^{3} b_{17}+\alpha^{2} b_{18}+\alpha b_{19}+b_{20} \tag{3.19}
\end{align*}
$$

Notice that the coefficients of $B_{1}, B_{2}$ and $B_{3}$ in (3.16) originate from the MDS encoding vectors. Hence, if we assume that $B_{2}$ is given, then the MDS property implies that (3.16) has a unique solution for $B_{1}$ and $B_{3}$. Moreover, since $B_{1}$ and $B_{3}$ have unique polynomial representations in $G F(32)$ of degree at most 4, then for given values of $B_{1}$ and $B_{3},(\mathrm{~A} .6)$ and (3.19) have at most one solution for $b_{3}, b_{4}, b_{5}, b_{16}, b_{17}, b_{18}, b_{19}$ and $b_{20}$. Therefore, an upper bound on $\gamma$ is given by the number of possible choices of $B_{2}$, i.e., $\gamma \leq 2^{2}=4$.

The analysis in Example A. 1 can be generalized for messages of any length $k$. Assume without loss of generality that $\delta=w$ deletions occur. Then, in general, the analysis yields $2 w$ "free" bits and $k-2 w$ "fixed" bits. Similar to (3.16), the $2 w$ "free" bits are constrained by a system of 2 linear equations in $G F(q)$. Note that this system of 2 linear equations in $G F(q)$ does not necessarily have exactly 2 variables. For instance, in Example A.1, we had 3 variables $\left(B_{1}, B_{2}, B_{3}\right)$ in the 2 equations. This happens because of the shift caused by the deletions which could lead to the $2 w$ "free" spanning up to 3 blocks, resulting in an additional symbol that is multiplied by a different MDS encoding coefficient. Therefore, since the difference between the number of symbols and number of equations is at most one, then the MDS property implies
that the number of solutions of the system of equations is at most $2^{w}$, i.e., $\gamma \leq 2^{w}$. Since $w \leq \ell$ from (3.1), the upper bound in (3.14) becomes

$$
\begin{equation*}
\operatorname{Pr}\left(\mathcal{Y}_{i}=\mathbf{Y} \mid W, p_{1}, p_{2}\right) \leq \frac{2^{\ell}}{2^{k-2 \ell}}=\frac{1}{2^{k-3 \ell}} \tag{3.20}
\end{equation*}
$$

The bound in (3.20) does not depend on the locations of the localized deletions and holds for an arbitrary window position $W$. Therefore, the upper bound on the probability of decoding failure in (3.13) holds for any window location that is chosen independently of the codeword. Moreover, for any given distribution on the window location (like the uniform distribution for example), we can apply the law of total probability and use the result from (3.20) to get

$$
\begin{equation*}
\operatorname{Pr}\left(\mathcal{Y}_{\boldsymbol{i}}=\mathbf{Y} \mid p_{1}, p_{2}\right) \leq \frac{1}{2^{k-3 \ell}} \tag{3.21}
\end{equation*}
$$

### 3.9 Conclusion

In this chapter, we introduced new explicit codes that can correct deletions that are localized within single or multiple windows in the codeword. These codes have several desirable properties such as: systematic, low redundancy, asymptotically optimal rate, efficient encoding and decoding, and low probability of decoding failure. We demonstrated these properties through our theoretical analysis and validated them through numerical simulations. Deriving fundamental limits for the problem of correcting localized deletions is an open problem and is one of the main future directions to consider. Another interesting direction is to apply these codes to file synchronization and compare their performance to other baseline algorithms such as rsync.

## Chapter 4

## Codes for Trace Reconstruction

### 4.1 Introduction

Motivated by applications to DNA-based storage systems, we introduce in this chapter novel codes for coded trace reconstruction. In coded trace reconstruction, the goal is to recover a coded sequence from multiple erroneous traces (copies) of it. The problem of reconstructing a sequence from multiple traces arises naturally when DNA is sequenced with nanopores, resulting in multiple erroneous copies of the stored data. We are interested in constructing efficient codes that enable reconstructing a sequence from a small number of traces in order to ensure reliability in such storage systems.

In this chapter, we focus on binary sequences affected by a small number of deletions as a first step towards designing codes that are robust against the various types of errors that are experienced in DNA-based storage such as deletions, insertions, substitutions, etc. We consider the random deletion model where each bit is deleted independently with probability $p=\Theta(1 / n)$. In this regime, the optimal redundancy from the coding theoretic perspective is $\Theta(\log n)$. We propose a construction which introduces a constant redundancy, i.e., $\mathcal{O}(1)$, per trace, that allows the reconstruction of the sequence from a small number of traces. Namely, our contributions are the following.

1. We construct novel codes with $\mathcal{O}(1)$ redundancy, that can efficiently reconstruct a binary sequence of length $n$ from $\mathcal{O}(1)$ traces, in the regime where the probability of a bit being deleted is $p=\Theta(1 / n)$. Our codes have a lower redundancy compared to the coded trace reconstruction scheme presented in [58], which has $\Theta(\log n)$ redundancy in a similar regime where the number of deletions in the codeword is fixed.
2. A main ingredient of our construction is designing a set of delimiter bits that enable detecting the number of deletions in certain parts of the codeword. Designing efficient codes for detecting deletions in the setting of concatenated codes is a problem of independent interest that has not received attention in the literature.
3. We implement our codes and provide numerical results on the performance of these codes. The results show that the codes have a low probability of error for a small number of traces. The results also show that this probability of error decreases as the blocklength $n$ increases.

The chapter is organized as follows. In Section 4.2, we introduce the deletion model and the notation used in this chapter. We present our code construction in Section 4.3 and explain the main ingredients of this construction. We state and discuss our theoretical results in Section 4.4.1. The simulation results are given in Section 4.4.2.

### 4.2 Preliminaries

We consider the following deletion model. For a given input sequence $x=\left\{x_{1}, x_{2}, \ldots, x_{n}\right\} \in$ $\{0,1\}^{n}$ a deletion probability $p=\Theta(1 / n)=k_{1} / n$ for some constant $k_{1}>0$, and an integer $t$, the channel returns $t$ traces of $x$. Each trace of $x$ is obtained by sending $x$ through a deletion channel with deletion probability $p$, i.e., the deletion channel deletes each bit $x_{i}$ of $x, i=1, \ldots, n$, independently with probability $p$, and outputs a subsequence of $x$ containing all bits of $x$ that were not deleted in order. The $t$ traces are independent and identically distributed as outputs of the deletion channel for input $x$. Let $a$ and $b$ be two binary sequences, we use the notation
$\langle a, b\rangle$ to refer to the concatenation of the two sequences $a$ and $b$.

### 4.3 Code Construction

In this section, we present our code construction and explain the main ideas behind the techniques that we use.

### 4.3.1 Overview

The high-level idea of our code construction is the following. We construct a codeword $x \in$ $\{0,1\}^{n}$, by concatenating $k_{2}$ codewords each of length $\ell=n / k_{2}$, where $k_{2} \in \mathbb{Z}^{+}$is a parameter of the code. Let $x^{a} \in\{0,1\}^{\ell}, a \in\left\{1, \ldots, k_{2}\right\}$, be the concatenated codewords which form the codeword $x=\left\langle x^{1}, x^{2}, \ldots, x^{k_{2}}\right\rangle \in\{0,1\}^{n}$. Henceforth, we use the term blocks to refer to $x^{1}, x^{2}, \ldots, x^{k_{2}}$. We construct each block in $x$ such that it satisfies the following two properties. First, each block has a small number of fixed bits in the beginning and in the end of the block, which we call delimiter bits. Second, the blocks satisfy a run-length-limited constraint, i.e., the length of each run of 0's or 1's is limited by a maximum value.

The goal of introducing the delimiter bits is to detect the deletions in each block, and consequently recover the block boundaries at the decoder, in order to obtain the traces corresponding to each block. This allows us to subdivide the trace reconstruction problem into smaller subproblems. This subdivision can serve several purposes such as decreasing the probability of decoding error for a given number of traces, or decreasing the number of traces needed to achieve a given probability of error. We discuss the importance of subdividing the problem in Section 4.3.5.

To reconstruct $x$ at the decoder, we use the following approach. The decoder obtains $t$ independent traces of $x$ resulting from $t$ independent deletion channels. To decode $x$, the decoder first uses the delimiter bits to recover the block boundaries of each trace, as we explain in Section 4.3.3. The decoder then attempts to recover each block by looking for a deletion-free copy
of that block among the traces. If a deletion-free copy exists, the block is successfully decoded. Otherwise, the decoder uses the Bitwise Majority Alignment (BMA) algorithm explained in Section 4.3.2, to reconstruct each block from its corresponding traces that are affected by deletions. To finalize the decoding, the recovered blocks are concatenated in order to reconstruct $x$.

### 4.3.2 Bitwise Majority Alignment

The bitwise majority alignment (BMA) algorithm was first introduced in [49] as a reconstruction algorithm for uncoded trace reconstruction. In our scheme, we use the BMA algorithm to reconstruct the blocks in the case where no deletion-free copy of that block is available among the traces.

The input of the algorithm is a $t \times \ell$ binary matrix which consists of the $t$ traces corresponding to a certain block. Since the length of some of the received traces may be smaller than $\ell$ due to the deletions, the traces in the input of the algorithm are padded to length $\ell$ by adding a special character (other than 0 or 1). The main idea of the algorithm follows a majority voting approach, where each bit is reconstructed based on its value in the majority of the traces. Namely, for each trace, a pointer is initialized to the leftmost bit, and the value of the bit in the reconstructed sequence is decided based on the votes of the majority of the pointers. Then, the pointers corresponding to the traces that voted with the majority are incremented by one, while other pointers remain at the same bit position. Thus, the reconstruction process scans the bits of the traces from left to right, and the pointers may possibly be pointing to different bit positions at different states of the algorithm. In the end, the algorithm outputs a single reconstructed sequence of length $\ell$ bits.

Let $q(j)$ be the pointer corresponding to trace $j, j=1, \ldots, t . q(j) \in\{1, \ldots, \ell\}$, where $q(j)=i, i=1, \ldots, \ell$, means that the pointer corresponding to trace $j$ is pointing to the bit at position $i$. The detailed algorithm is given in Algorithm 1.

The authors in [49] showed that the BMA algorithm given in Algorithm 1 can reconstruct random sequences with high probability, i.e., probability of error vanishes asymptotically with

```
Algorithm 1: Bitwise Majority Alignment
    input : A binary matrix \(T\) of size \(t \times \ell\)
    output: A binary sequence \(y\) of size \(\ell\)
    Let \(q(j) \leftarrow 1\) for all \(j=1, \ldots, t\);
    for \(i \leftarrow 1\) to \(\ell\) do
        Let \(b\) be the majority over all \(j\) for \(T(j, c(j))\);
        \(y(i) \leftarrow b\);
        for \(j \leftarrow 1\) to \(t\) do
                if \(q(j)==b\) then
                    \(q(j) \leftarrow q(j)+1 ;\)
            end
        end
    end
```

the length of the sequence $n$. However, the BMA algorithm can have a poor performance for some arbitrary sequences, particularly sequences that have long runs of 0's or 1's. For instance, consider a sequence that starts with a long run of 0 's. Due to the deletions, the number of 0 's observed for this run differs from one trace to another. While scanning from left to right, at some point before the end of this run, the majority of the traces could vote for a 1 . This will lead to splitting the run resulting in an erroneous reconstruction. Nevertheless, in the case where the lengths of the runs are limited to a maximum value of $\sqrt{n}$, the authors in [49] proved that the BMA algorithm can reconstruct an arbitrary uncoded sequence of length $n$ with high probability, using a constant number of traces $\mathcal{O}(1)$, when the deletion probability satisfies $p=\mathcal{O}(1 / \sqrt{n})$. To this end, we restrict our code to run-length-limited sequences where the length of any run of 0 's or 1 's is upper bounded by $\sqrt{n}$.

### 4.3.3 Delimiter Bits

As previously mentioned, the goal of the delimiter bits is to detect the number of deletions in each block, and in turn enable the recovery of the blocks in each trace after the sequence $x$ passes through the deletion channels. In this section, we present a code that can detect up to 1 deletion in each block. Note that detecting the number of deletions per block in the setting of concatenated codes is a problem of independent interest. It is also possible to construct codes (with lower rate) that can detect more than one deletion per block. To this end, we also present
in this section a code construction that can detect up to 2 deletions per block.
Define the code

$$
\begin{equation*}
\mathcal{A}_{1}^{\ell} \triangleq\left\{x \in\{0,1\}^{\ell} \mid\left(x_{1}, x_{2}, x_{\ell-1}, x_{\ell}\right)=(0,0,0,1)\right\} \tag{4.1}
\end{equation*}
$$

where $\ell=n / k_{2}$ and $k_{2} \in \mathbb{Z}^{+}$. We construct the following code $\mathcal{D}_{1}$ of dimension $n$ by concatenating $k_{2} \in \mathbb{Z}^{+}$codewords of $\mathcal{A}_{1}^{\ell}$, where $k_{2}$ is a parameter of the code, i.e.,

$$
\begin{equation*}
\mathcal{D}_{1} \triangleq\left\{\left\langle x^{1}, x^{2}, \ldots, x^{k_{2}}\right\rangle \mid x^{a} \in \mathcal{A}_{1}^{\ell}, \forall a \in\left\{1, \ldots, k_{2}\right\}\right\} \tag{4.2}
\end{equation*}
$$

Lemma 4.1. The code $\mathcal{D}_{1}$ can detect up to 1 deletion in each of the $k_{2}$ blocks.

Proof. To show that the code $\mathcal{D}_{1}$ can detect up to one deletion per block, we go over all the possible observations in positions $\ell-1$ and $\ell$ after the sequence $x \in \mathcal{D}_{1}$ passes through the deletion channel. Based on these observations, we form a decision rule that allows us to detect the number of deletions in the first block. As a result, we can determine the boundary of the first block and then apply the same decision rule to detect the deletions in the remaining blocks by operating on a block-by-block basis. The possible observations are shown below.

1. No deletions: In this case, in positions $\ell-1$ and $\ell$, we observe 01 .

## 2. One deletion:

- Deletion in the first $\ell-2$ bits: In this case, in positions $\ell-1$ and $\ell$, we observe 10. Note that the reason why the bit observed in position $\ell$ is 0 , follows from the code construction which fixes the first 2 bits in the second block to 00. So given the assumption that at most one bit gets deleted in the second block, we always observe a 0 in position $\ell$.
- Deletion in the last 2 bits: In this case, in positions $\ell-1$ and $\ell$, we observe 00 . Here, a 0 is observed in position $\ell$ for the same reason discussed in the previous case.

Based on these observations, we can form the following decision table.

| Observation |  | Decision |
| :---: | :---: | :---: |
| position $\ell-1$ | position $\ell$ |  |
| 0 | 1 | no deletions |
| 1 | 0 | 1 deletion in first $\ell-1$ bits |
| 0 | 0 | 1 deletion in the last bit |

Figure 4.1: The table shows the observations in positions $\ell-1$ and $\ell$, at the decoder if the first block of $x \in \mathcal{D}_{1}$ is affected by at most one deletion after it passes through the deletion channel. The corresponding decision rules are also given.

It is easy to see from Figure 4.1 that the observations allow us to detect the number of deletions in each block with zero-error, given the assumption that each block is affected by at most one deletion.

Remark 4.2. Notice from Figure 4.1 that in some cases, we obtain some information in addition to detecting the number of deletions in the block. For instance, in the case where we observe 00 in positions $\ell-1$ and $\ell$, we can deduce that a single deletion has affected the last delimiter bit, given the assumption that the block was affected by at most one deletion. Observing 00 in positions $\ell-1$ and $\ell$ could also indicate that the block was affected by two deletions, which is probable in our random deletion model setting. In fact, the probability that a certain block was affected by two deletions is much higher than the probability that a single deletion affected the last delimiter, for interesting choices of the code parameters. Therefore, $\mathcal{D}_{1}$ can also detect 2 deletions in a block for most of the cases. In practice, we can exploit this additional information to improve the performance of the reconstruction algorithm.

Next, we provide a code construction that can detect up to 2 deletions per block. Define the following code

$$
\begin{equation*}
\mathcal{A}_{2}^{\ell} \triangleq\left\{x \in\{0,1\}^{\ell} \mid\left(x_{1}, x_{2}, x_{3}, x_{\ell-2}, x_{\ell-1}, x_{\ell}\right)=(0,0,0,0,1,1)\right\} \tag{4.3}
\end{equation*}
$$

Similar to the code $\mathcal{D}_{1}$, we construct the code $\mathcal{D}_{2}$ of dimension $n$ by concatenating $k_{2}$ codewords
of $\mathcal{A}_{2}^{\ell}$, i.e.,

$$
\begin{equation*}
\mathcal{D}_{2} \triangleq\left\{\left\langle x^{1}, x^{2}, \ldots, x^{k_{2}}\right\rangle \mid x^{a} \in \mathcal{A}_{2}^{\ell}, \forall a \in\left\{1, \ldots, k_{2}\right\}\right\} \tag{4.4}
\end{equation*}
$$

Lemma 4.3. The code $\mathcal{D}_{2}$ can detect up to 2 deletions in each of the $k_{2}$ blocks.

Proof. Similar to the previous analysis for $\mathcal{D}_{1}$, the decision table in Figure 4.2 follows from listing all the possible observations of the bits in positions $\ell-2, \ell-1$, and $\ell$ after the sequence $x \in \mathcal{D}_{2}$ passes through the deletion channel.

| Observation |  |  | Decision |
| :---: | :---: | :---: | :---: |
| position $\ell-2$ | position $\ell-1$ | position $\ell$ |  |
| 0 | 1 | 1 | no deletions |
| 1 | 1 | 0 | 1 deletion in first $\ell-2$ bits |
| 0 | 1 | 0 | 1 deletion in the last two bits |
| 0 | 0 | $\times$ | 2 deletions in the last two bits |
| 1 | 0 | $\times$ | 2 deletions |

Figure 4.2: The table shows the observations in positions $\ell-2, \ell-1 \ell$, at the decoder if the first block if $x \in \mathcal{D}_{2}$ is affected by at most two deletions after it passes through the deletion channel. $\times$ refers to an irrelevant entry. The corresponding decision rules are also given.

It is also possible to generalize the constructions above to codes that can detect more than 2 deletions per block, which is a problem of independent interest.

### 4.3.4 Code Construction

Next, we formally present two novel code constructions for coded trace reconstruction. Let $\mathcal{L}^{\ell}(f(n)) \subseteq\{0,1\}^{\ell}$ be the set of run-length-limited binary sequences of length $\ell=n / k_{2}$ where the length of any run of 0 's or 1's in a sequence $x \in \mathcal{L}^{\ell}(f(n))$ is at most $f(n)$.

Consider the following two code constructions:

$$
\begin{align*}
& \mathcal{C}_{1} \triangleq\left\{\left\langle x^{1}, x^{2}, \ldots, x^{k_{2}}\right\rangle \mid x^{a} \in \mathcal{A}_{1}^{\ell}, x^{a} \in \mathcal{L}^{\ell}(\sqrt{n}), \forall a \in\left\{1, \ldots, k_{2}\right\}\right\},  \tag{4.5}\\
& \mathcal{C}_{2} \triangleq\left\{\left\langle x^{1}, x^{2}, \ldots, x^{k_{2}}\right\rangle \mid x^{a} \in \mathcal{A}_{2}^{\ell}, x^{a} \in \mathcal{L}^{\ell}(\sqrt{n}), \forall a \in\left\{1, \ldots, k_{2}\right\}\right\}, \tag{4.6}
\end{align*}
$$

where $k_{2} \in \mathbb{Z}^{+}$is a parameter of the code. The steps of the reconstruction algorithm for a sequence $x \in \mathcal{C}_{1}$ or $x \in \mathcal{C}_{2}$ are the following:

1. Recover the block boundaries in all $t$ traces based on the decision rules given in Figure 4.1 or 4.2.
2. Search for a deletion-free copy of each block among the $t$ traces. If a deletion-free copy of a certain block exists, the block is reconstructed. Otherwise, apply the BMA algorithm (Algorithm 1) to reconstruct the block.
3. Concatenate the reconstructed blocks to obtain $x$.

### 4.3.5 Discussion

The proposed codes enable subdividing the trace reconstruction problem into smaller subproblems by allowing the decoder to recover the traces corresponding to each block. This subdivision can serve many purposes such as decreasing the probability of error for a given number of traces, or decreasing the number of traces for a given target probability of error.

To understand the intuition behind the subdivision, consider the simple trace reconstruction algorithm which attempts to reconstruct the sequence by finding a deletion-free copy of it among the traces. In the uncoded setting with a small probability of deletion, we expect to observe a small number of deletions per trace, however we also expect that many parts of the trace will be deletion-free. Hence, if we use a code that enables subdividing the problem, then we have a higher chance of reconstructing the sequence by finding one deletion-free copy of each block among the traces.

For example, consider the setting where the decoder has $t=2$ traces of a binary sequence $x$ of length $n$. Suppose that the first trace $y_{1}$ is affected by a single deletion at position $i$ where $i<n / 2$; and the second trace $y_{2}$ is affected by a single deletion at position $j$ where $j>n / 2$.

Therefore, the decoder has

$$
\begin{aligned}
& y_{1}=\left(x_{1}, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{n / 2}, \ldots, x_{n}\right), \\
& y_{2}=\left(x_{1}, \ldots, x_{n / 2}, \ldots, x_{j-1}, x_{j+1}, \ldots, x_{n}\right)
\end{aligned}
$$

If $x$ is uncoded, the reconstruction is unsuccessful since there is no deletion-free copy of $x$ in the 2 traces $y_{1}$ and $y_{2}$. Whereas, if $x \in \mathcal{D}_{1}$ (defined in (4.2)) with $k_{2}=2$ and $\ell=n / 2$, then $x$ can be successfully reconstructed since we have a deletion-free copy of the first block in the second trace, and a deletion-free copy of the second block in the first trace.

To further understand the effect of code, we analyze the probability of error of the simple reconstruction algorithm that searches for a deletion-free copy, in both the uncoded and coded settings. Let $P_{\text {uncoded }}$ and $P_{\text {coded }}$ be the probability of error for the uncoded and coded settings, respectively. For the coded case we consider the code $\mathcal{D}_{1}$, and we assume for the sake of simplicity that the subdivision is successful, i.e., no block has more than 1 deletion.

The probability of error in the uncoded case is the probability that all the $t$ traces of the sequence $x \in\{0,1\}^{n}$ have at least one deletion. Hence, the probability of error is given by

$$
\begin{equation*}
P_{\text {uncoded }}=\left[1-\left(1-\frac{k_{1}}{n}\right)^{n}\right]^{t} \tag{4.7}
\end{equation*}
$$

As for the coded case, the reconstruction of $x \in \mathcal{D}_{1}$ is successful in the event where the reconstruction of all the $k_{2}$ blocks is successful, where each block is of size $\ell=n / k_{2}$. Therefore, the probability of error is given by

$$
\begin{equation*}
P_{\text {coded }}=1-\left[1-\left(1-\left(1-\frac{k_{1}}{n}\right)^{\ell}\right)^{t}\right]^{k_{2}} \tag{4.8}
\end{equation*}
$$

In Figure 4.3 we compare the two quantities $P_{\text {uncoded }}$ and $P_{\text {coded }}$ for multiple values of the number of traces $t$, for $n=1000, k_{1}=3, k_{2}=5$ and $\ell=200$.

The comparison shows that the code allows achieving a lower probability of error for the


Figure 4.3: Comparing the probability of errors for the uncoded and coded cases $P_{\text {uncoded }}$ and $P_{\text {coded }}$, for $n=1000, k_{1}=3, k_{2}=5$ and $\ell=200$.
same number of traces. Moreover, the gain in terms of probability of error increases as the number of traces $t$ increases.

### 4.4 Results

In this section, we state our main results formally, and also provide simulation results on the performance of our codes.

### 4.4.1 Theoretical Results

Next, we state and discuss our main theoretical results. Theorem 4.4 gives a lower bound on the rates of the codes $\mathcal{C}_{1}$ and $\mathcal{C}_{2}$, defined in (4.5) and (4.6), respectively. The results show that these codes have a constant redundancy if we fix the code parameter $k_{2}$ with respect to $n$. Theorem 4.5 shows that under the assumption that the block boundaries are recovered successfully at the decoder, the codes $\mathcal{C}_{1}$ and $\mathcal{C}_{2}$ with constant redundancy, can be efficiently reconstructed from a constant number of traces, with high probability. We also discuss the probability of the event where the block boundaries are not successfully recovered at the decoder, and show that this probability can be made sufficiently small for appropriate choices of the code parameters.

Theorem 4.4. The rates of the codes $\mathcal{C}_{1} \subseteq\{0,1\}^{n}$ and $\mathcal{C}_{2} \subseteq\{0,1\}^{n}$ satisfy

$$
R_{1}\left(k_{2}\right) \geq 1-\frac{4 k_{2}+o(1)}{n}
$$

and

$$
R_{2}\left(k_{2}\right) \geq 1-\frac{6 k_{2}+o(1)}{n}
$$

respectively, where $k_{2} \in \mathbb{Z}^{+}$is a code parameter.

Proof. The rate of $\mathcal{C}_{1}$ is given by

$$
\begin{equation*}
R_{1}\left(k_{2}\right)=\frac{\log _{2}\left|\mathcal{C}_{1}\right|}{n} \tag{4.9}
\end{equation*}
$$

The first constraint of the code $\mathcal{C}_{1}$ fixes the first and last two bits of each of the $k_{2}$ blocks, which results in a degree of freedom that is $n-4 k_{2}$. Furthermore, the second constraint limits the length of any run to a maximum of $\sqrt{n}$. Let $\mathcal{L}^{n}(\sqrt{n})$ be the set of run-length-limited binary sequences of length $\ell=n$ where the length of any run of 0 's or 1 's in a sequence $x \in \mathcal{L}^{n}(\sqrt{n})$ is at most $\sqrt{n}$. Therefore,

$$
\begin{equation*}
\left|\mathcal{C}_{1}\right| \geq 2^{n-4 k_{2}}-\left|\overline{\mathcal{L}^{n}(\sqrt{n})}\right| \tag{4.10}
\end{equation*}
$$

Let $L_{n}$ be a random variable that denotes the length of the longest run in a binary sequence of length $n$, where the sequences are chosen uniformly at random. We are interested in computing an upper bound on the probability $\operatorname{Pr}\left(L_{n}>\sqrt{n}\right)$. By the union bound, it is enough to require that every window of size $\sqrt{n}$ is not all 0's or all 1's. Hence, we get

$$
\begin{equation*}
\operatorname{Pr}\left(L_{n}>\sqrt{n}\right) \leq n \cdot \frac{2}{2^{\sqrt{n}}} \tag{4.11}
\end{equation*}
$$

Therefore,

$$
\begin{equation*}
\left|\overline{\mathcal{L}^{n}(\sqrt{n})}\right| \leq 2^{n} \cdot \frac{2 n}{2^{\sqrt{n}}} \tag{4.12}
\end{equation*}
$$

By substituting (4.12) in (4.10) we get

$$
\begin{equation*}
\left|\mathcal{C}_{1}\right| \geq 2^{n-4 k_{2}}\left(1-2^{4 k_{2}-1} \cdot \frac{n}{2^{\sqrt{n}}}\right) \tag{4.13}
\end{equation*}
$$

Hence,

$$
\begin{equation*}
R_{1}\left(k_{2}\right) \geq 1-\frac{4 k_{2}}{n}-\frac{1}{n} \log _{2}\left(\frac{2^{\sqrt{n}}}{2^{\sqrt{n}}-2^{4 k_{2}-1} n}\right) \tag{4.14}
\end{equation*}
$$

Furthermore, since

$$
\begin{equation*}
\lim _{n \rightarrow+\infty} \frac{2^{\sqrt{n}}}{2^{\sqrt{n}}-2^{4 k_{2}-1} n}=1 \tag{4.15}
\end{equation*}
$$

we get

$$
\begin{equation*}
R_{1}\left(k_{2}\right)=1-\frac{4 k_{2}+o(1)}{n} \tag{4.16}
\end{equation*}
$$

The result on the rate of $\mathcal{C}_{2}$ follows from the same analysis above. The only difference is that the number of fixed bits in $\mathcal{C}_{2}$ is 6 bits per block.

Theorem 4.5. Assuming that the block boundaries are recovered successfully at the decoder, then for the random i.i.d. deletion model with deletion probability $p=\Theta(1 / n)$, a codeword $x \in \mathcal{C}_{1}$ or $x \in \mathcal{C}_{2}$ can be reconstructed with high probability, and in linear time, from $\mathcal{O}(1)$ traces, where the redundancy of the code is constant.

Proof. It follows from Theorem 4.4 that the redundancy of the codes $x \in \mathcal{C}_{1}$ and $x \in \mathcal{C}_{2}$ is constant for a fixed choice of the code parameter $k_{2}$ with respect to $n$. Moreover, we know from [49] that the probability of error of the BMA algorithm (Algorithm 1) is $\mathcal{O}(1 / n)$, for $p=\Theta(1 / n)$, a constant number of traces, and an arbitrary sequence that does not have runs of length greater than $\sqrt{n}$. The second constraint in the codes guarantees that no codeword has a run of length greater than $\sqrt{n}$. Therefore, assuming that the block boundaries are recovered successfully at the decoder using the delimiter bits, the reconstruction is successful with high probability, where the probability is over the randomness of the deletion process. Furthermore, the time complexity of the BMA algorithm is linear in the size of input matrix $t \times \ell$, since every
trace is scanned once in the reconstruction process.

In our code construction, we have an additional potential cause of an error, which is the event where the boundaries of the blocks are not recovered successfully at the decoder, i.e., more than one deletion is experienced in one of the $k_{2}$ blocks in the case of $\mathcal{C}_{1}$, or more than two deletions in the case $\mathcal{C}_{2}$. The number of deletions per block follows a $\operatorname{Binomial}\left(n / k_{2}, k_{1} / n\right)$ distribution. Let $X \sim \operatorname{Binomial}\left(n / k_{2}, k_{1} / n\right)$ be the random variable that represents the number of deletions in a certain block. The probability that the number of deletions in a certain block is greater than $m \in \mathbb{N}$ is

$$
\begin{equation*}
\operatorname{Pr}(X \geq m)=\sum_{i=m}^{n / k_{2}}\left(\frac{k_{1}}{n}\right)^{m}\left(1-\frac{k_{1}}{n}\right)^{\frac{n}{k_{2}}-m} \tag{4.17}
\end{equation*}
$$

The average number of deletions per block is $E[X]=k_{1} / k_{2}$ and the variance is $\operatorname{Var}[X]=$ $k_{1} / k_{2}\left(1-k_{1} / n\right)$. In Lemma 4.6, we bound the probability in (4.17). The proof of this lemma is given in Appendix C.

Lemma 4.6. For $k_{2}>k_{1}$ and $m \geq 1$, we have

$$
\begin{equation*}
\operatorname{Pr}(X \geq m)<\left(\frac{e k_{1}}{m k_{2}}\right)^{m} e^{-\frac{k_{1}}{k_{2}}} \tag{4.18}
\end{equation*}
$$

The code $\mathcal{C}_{1}$ can detect up to one deletion per block, so we are interested in $\operatorname{Pr}(X \geq 2)$, which can be made sufficiently small by choosing $k_{2}$ large enough with respect to $k_{1}$ based on Lemma 4.6. Note that this is intuitive since if $k_{2}$ is large with respect to $k_{1}$, then the average number of deletions per block $k_{1} / k_{2}$ is much smaller than 1 . Here, it is noteworthy to recall that $\mathcal{C}_{1}$ can also detect most 2 deletions per block (Remark 4.2).

The probability of the event where the block boundaries are not recovered successfully can be decreased further by using the code $\mathcal{C}_{2}$ which can detect up to 3 deletions per block. Notice from Lemma 4.6, that the bound on the probability decreases exponentially with $m$. Hence, we expect a significant performance enhancement by using $\mathcal{C}_{2}$, at the expense of a higher redundancy. Another option is to use a superconstant redundancy by choosing $k_{2}=\omega(1)$, which makes the
probability in (4.18) vanish asymptotically.
Note that in practice, based on the decision rules in Figures 4.1 and 4.2, one can sometimes detect that the process of recovering the block boundaries of a certain trace was erroneous. In this case, one can simply drop that trace, and run the reconstruction algorithm on the remaining traces. In the next section, we provide numerical results which show that our codes are efficient in practice and can achieve a low probability of error with a small number of redundant bits.

### 4.4.2 Simulation Results

In this section, we provide simulation results on the performance of our codes. Note that our theoretical analysis on the probability of error in the previous section applies for arbitrary sequences which satisfy the constraints in $\operatorname{codes} \mathcal{C}_{1}$ and $\mathcal{C}_{2}$. In our numerical study, the empirical probability of error is averaged over several trials, where in each trial the information bits are chosen uniformly at random. The probability that a random sequence has a run of size greater than $\sqrt{n}$ is bounded in (4.11), and is very low. For this reason, we drop the run-length constraint in our numerical study and adopt the following simple encoding algorithm. To encode a certain information sequence $u$ into a codeword $x \in \mathcal{C}_{1}$, we first divide $u$ into $k_{2}$ binary blocks of equal sizes (we pad the last block if necessary). For each block, we insert two zeros (00) in the beginning, and a zero followed by a one (01) in the end. Then, we concatenate the binary blocks to obtain a codeword $x \in \mathcal{C}_{1}$. The encoding for $\mathcal{C}_{2}$ can be obtained similarly.

Figure 4.4 shows the empirical probability of error of the codes $\mathcal{C}_{1}$ and $\mathcal{C}_{2}$ for different values of the number of traces $t$. The length of the codeword is fixed to $n=10^{4}$, the code parameters are $k_{1}=5$ (deletion probability $p=k_{1} / n=5 / n$ ), $k_{2}=10$ (the codeword is divided into 10 blocks). Hence, the number of redundant bits is 40 bits for $\mathcal{C}_{1}$, and 60 bits for $\mathcal{C}_{2}$. The probability of error is averaged over $10^{4}$ trials. The results show that the probability of error of both codes decreases exponentially (roughly) with the number of traces; moreover, the probability of error of $\mathcal{C}_{2}$ is much smaller than that of $\mathcal{C}_{1}$. For $t>6$, we could not detect any error for $\mathcal{C}_{2}$ within the $10^{4}$ trials.


Figure 4.4: Empirical probability of error of the codes $\mathcal{C}_{1}$ and $\mathcal{C}_{2}$ for multiple values of the number of traces $t$. The length of the codeword is $n=10^{4}$ bits, and the code parameters are $k_{2}=10$ and $k_{1}=5$. The results are averaged over $10^{4}$ trials.

Figure 4.5 shows the empirical probability of error for different values of $n$. The number of traces is fixed to $t=4$, and the code parameters are $k_{1}=5, k_{2}=10$. The number of redundant bits remain fixed as we vary $n$, where the redundancy of $\mathcal{C}_{1}$ is 40 bits and the redundancy of $\mathcal{C}_{2}$ is 60 bits. The probability of error is averaged over $10^{4}$ trials. The results show that the probability of error of both codes decreases with $n$.


Figure 4.5: Empirical probability of error of the $\operatorname{codes} \mathcal{C}_{1}$ and $\mathcal{C}_{2}$ for multiple values of the length of the codeword $n$. The length of the codeword is $t=4$ bits, and the code parameters are $k_{2}=10$ and $k_{1}=5$. The results are averaged over $10^{4}$ trials.


Figure 4.6: Empirical probability of error of the code $\mathcal{C}_{1}$ with redundancy $4 k_{2}$, for multiple values of the code parameter $k_{2}$. The length of the codeword is $n=10^{4}$ bits, the number of traces is $t=6$, and the deletion probability is $p=k_{1} / n=5 / n$. The results are averaged over $10^{4}$ trials.


Figure 4.7: Empirical probability of error of the code $\mathcal{C}_{1}$ with redundancy $4 k_{2}$, for multiple values of the code parameter $k_{2}$. The length of the codeword is $n=10^{4}$ bits, the number of traces is $t=4$, and the deletion probability is $p=k_{1} / n=5 / n$. The results are averaged over $10^{4}$ trials.

Figure 4.6 shows the empirical probability of error of $\mathcal{C}_{1}$ for $n=10^{4}, k_{1}=5$, and for different values of the code parameter $k_{2}$. The number of traces is fixed to $t=6$. Note that the redundancy of $\mathcal{C}_{1}$ is $4 k_{2}$, which increases linearly with $k_{2}$. The probability of error is averaged
over $10^{4}$ trials. The results show that increasing the redundancy results in a significant decrease in the probability of error.

Figure 4.7 shows the empirical probability of error of $\mathcal{C}_{1}$ and $\mathcal{C}_{2}$ for $n=10^{4}, k_{1}=5, t=4$, and for different fixed values of the redundancy. The value of the code parameter $k_{2}$ for $\mathcal{C}_{1}$ and $\mathcal{C}_{2}$ are determined accordingly such that both codes have the same redundancy. The probability of error is averaged over $10^{4}$ trials. The results show that for the particular setting considered in Figure 4.7, $\mathcal{C}_{2}$ has a better performance than $\mathcal{C}_{1}$ for the same redundancy values. This observation further motivates the study of codes that can detect multiple deletions per block.

### 4.5 Conclusion

In this chapter, we introduced novel codes for trace reconstruction, in the case where the traces are affected by deletions. We studied the random deletion channel model where each bit is deleted independently with probability $p=\Theta(1 / n)$. In this setting, using a deletion correcting code would optimally require $\Theta(\log n)$ redundancy. The codes that we introduced for trace reconstruction have a constant redundancy (i.e., $\mathcal{O}(1)$ ), and enable reconstructing a sequence efficiently from a constant number of traces. We analyzed the performance of our codes both theoretically and numerically. In future work, we plan on generalizing our constructions to deal with other types of errors that are experienced in DNA-based storage systems, such as insertions and substitutions. Constructing efficient codes for the broad regime where $p=o(1)$ is an open problem. The problem of constructing optimal codes that can detect deletions in the setting of concatenated codes is also an interesting direction for future research.

## Chapter 5

## Conclusion

Deletions and insertions are the most notorious types of errors in error control coding. In fact, determining the information-theoretic capacity of the deletion channel remains to this day an open problem. There has also been lots of work since the 1960's on designing codes for correcting deletions and insertions. Constructing efficient codes for these errors proved to be a very challenging problem over the years - even for correcting only 2 deletions in a binary string. Most of the literature focuses on constructing codes that have zero-error. In this dissertation, motivated by applications to file synchronization and DNA-based storage, we focused on designing efficient codes for deletions and insertions that have a low probability of failure. We introduced novel codes for different deletion models: unrestricted deletions, localized deletions, and trace reconstruction.

In Chapter 2, we focused on the fundamental problem of constructing codes that can correct multiple unrestricted deletions (or insertions) in a binary codeword on length $n$. We introduced a new family of codes, that we called Guess \& Check (GC) codes, that can correct multiple deletions (or insertions), and have several desirable properties. GC codes can correct a constant number of $\delta$ deletions (or insertions), with high probability; and with a redundancy that is logarithmic in $k$, where $k$ is the length of the information message. Furthermore, these codes are explicit and systematic, which enables their application to remote file synchronization. We
provided deterministic polynomial time encoding and decoding algorithms for these codes. We also analyzed the list decoding performance of GC codes. We implemented these codes and validated our theoretical results by numerical simulations. Moreover, we applied these codes to file synchronization, and provided simulations results which highlight the gain obtained by using GC codes in this setting, compared to the state of the art synchronization algorithm. As a direction for future research, it is interesting to generalize our constructions to the regime where the number of deletions grows as a sub-linear function of $n$. It is also interesting to construct efficient systematic codes that can correct a mixed number of deletions and insertions.

In Chapter 3, we introduced and studied the model of localized deletions, where the positions deletions are restricted to certain parts (windows) of the codeword that are unknown a priori. This model is a generalization of the bursty model in which all the deletions must be consecutive. The localized model is particularly relevant to file synchronization, where it is often the case that large files are edited by deleting and inserting characters in a relatively small part of the file, resulting in a series of deletions and insertions that are localized within that part of the file. We designed novel efficient codes for this model which have the following properties. The codes are systematic and can correct, with high probability, $\delta \leq w$ deletions that are localized within a window of size $w=o(k)$ in the codeword. The redundancy of our codes is lower than that of other codes in the literature designed for the more restrictive bursty model. The rate of our codes is asymptotically optimal over the regime $w=o(k)$. The complexity of the encoding algorithm is near-linear, and the decoding complexity is quadratic. We implemented these codes and provided simulations results on their performance. We also generalized our construction to deal with deletions that are localized within multiple windows. For future work, it is interesting to generalize these codes so that they can deal with insertions as well as deletions. Deriving fundamental limits for correcting localized deletions is also an interesting open problem.

In Chapter 4, we studied the problem of coded trace reconstruction in the case where the traces are affected by random deletions. This problem has applications in DNA-based storage systems, where the process of sequencing DNA results in multiple erroneous copies of
the data. We introduce new codes that enhance reliability is such storage systems by allowing to reconstruct the coded data from few traces. We study the model where the traces are outputs of independent deletion channels that delete each bit independently with probability $p=\Theta(1 / n)$. Our codes have a constant redundancy and enable reconstructing a sequence efficiently from a small number of traces. We provided theoretical and numerical results on the performance of these codes. For future work, it is interesting to construct codes that can deal with other types of errors experienced in DNA-based storage systems, such as insertions and substitutions. Constructing efficient codes for the wide regime of $p=o(1)$ is also an interesting direction for future research.

## References

[1] R. Venkataramanan, V. N. Swamy, and K. Ramchandran, "Low-complexity interactive algorithms for synchronization from deletions, insertions, and substitutions," IEEE Transactions on Information Theory, vol. 61, no. 10, pp. 5670-5689, 2015.
[2] S. S. T. Yazdi and L. Dolecek, "A deterministic polynomial-time protocol for synchronizing from deletions," IEEE Transactions on Information Theory, vol. 60, no. 1, pp. 397-409, 2014.
[3] N. Ma, K. Ramchandran, and D. Tse, "Efficient file synchronization: A distributed source coding approach," in Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on. IEEE, 2011, pp. 583-587.
[4] S. El Rouayheb, S. Goparaju, H. M. Kiah, and O. Milenkovic, "Synchronization and deduplication in coded distributed storage networks," IEEE/ACM Transactions on Networking, vol. 24, no. 5, pp. 3056-3069, October 2016.
[5] R. Venkataramanan, H. Zhang, and K. Ramchandran, "Interactive low-complexity codes for synchronization from deletions and insertions," in 201048 th Annual Allerton Conference on Communication, Control, and Computing (Allerton), Sept 2010, pp. 1412-1419.
[6] F. Sala, C. Schoeny, N. BitouzÃl', and L. Dolecek, "Synchronizing files from a large number of insertions and deletions," IEEE Transactions on Communications, vol. 64, no. 6, pp. 2258-2273, June 2016.
[7] R. Gabrys, E. Yaakobi, and O. Milenkovic, "Codes in the damerau distance for DNA storage," in 2016 IEEE International Symposium on Information Theory (ISIT), July 2016, pp. 2644-2648.
[8] M. Cheraghchi, R. Gabrys, O. Milenkovic, and J. Ribeiro, "Coded trace reconstruction," IEEE Transactions on Information Theory, pp. 1-1, 2020.
[9] M. Mitzenmacher and E. Drinea, "A simple lower bound for the capacity of the deletion channel," IEEE Transactions on Information Theory, vol. 52, no. 10, pp. 4657-4660, Oct 2006.
[10] E. Drinea and M. Mitzenmacher, "On lower bounds for the capacity of deletion channels," IEEE Transactions on Information Theory, vol. 52, no. 10, pp. 4648-4657, Oct 2006.
[11] S. Diggavi, M. Mitzenmacher, and H. D. Pfister, "Capacity upper bounds for the deletion channel," in 2007 IEEE International Symposium on Information Theory, June 2007, pp. 1716-1720.
[12] R. Venkataramanan, S. Tatikonda, and K. Ramchandran, "Achievable rates for channels with deletions and insertions," IEEE Transactions on Information Theory, vol. 59, no. 11, pp. 6990-7013, Nov 2013.
[13] Y. Kanoria and A. Montanari, "Optimal coding for the binary deletion channel with small deletion probability," IEEE Transactions on Information Theory, vol. 59, no. 10, pp. 61926219, Oct 2013.
[14] M. Rahmati and T. M. Duman, "Upper bounds on the capacity of deletion channels using channel fragmentation," IEEE Transactions on Information Theory, vol. 61, no. 1, pp. 146-156, Jan 2015.
[15] M. Cheraghchi, "Capacity upper bounds for deletion-type channels," Journal of the ACM (JACM), vol. 66, no. 2, pp. 1-79, 2019.
[16] M. Mitzenmacher, "A survey of results for deletion channels and related synchronization channels," Probability Surveys, vol. 6, pp. 1-33, 2009.
[17] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals," in Soviet physics doklady, vol. 10, 1966, p. 707.
[18] R. Varshamov and G. TenengolâĂŹts, "Correction code for single asymmetric errors," Automat. Telemekh, vol. 26, no. 2, pp. 286-290, 1965.
[19] K. A. Abdel-Ghaffar and H. C. Ferreira, "Systematic encoding of the varshamov-tenengol'ts codes and the constantin-rao codes," IEEE Transactions on Information Theory, vol. 44, no. 1, pp. 340-345, 1998.
[20] N. J. Sloane, "On single-deletion-correcting codes," Codes and Designs, de Gruyter, Berlin, pp. 273-291, 2002.
[21] D. Cullina and N. Kiyavash, "An improvement to levenshtein's upper bound on the cardinality of deletion correcting codes," IEEE Transactions on Information Theory, vol. 60, no. 7, pp. 3862-3870, July 2014.
[22] A. S. Helberg and H. C. Ferreira, "On multiple insertion/deletion correcting codes," IEEE Transactions on Information Theory, vol. 48, no. 1, pp. 305-308, 2002.
[23] K. A. Abdel-Ghaffar, F. Paluncic, H. C. Ferreira, and W. A. Clarke, "On helberg's generalization of the levenshtein code for multiple deletion/insertion error correction," IEEE Transactions on Information Theory, vol. 58, no. 3, pp. 1804-1808, 2012.
[24] L. J. Schulman and D. Zuckerman, "Asymptotically good codes correcting insertions, deletions, and transpositions," IEEE transactions on information theory, vol. 45, no. 7, pp. 2552-2557, 1999.
[25] V. Guruswami and C. Wang, "Deletion codes in the high-noise and high-rate regimes," IEEE Transactions on Information Theory, vol. 63, no. 4, pp. 1961-1970, April 2017.
[26] V. Guruswami and R. Li, "Efficiently decodable insertion/deletion codes for high-noise and high-rate regimes," in 2016 IEEE International Symposium on Information Theory (ISIT), July 2016, pp. 620-624.
[27] —_, "Coding against deletions in oblivious and online models," arXiv preprint arXiv:1612.06335, 2016.
[28] J. Brakensiek, V. Guruswami, and S. Zbarsky, "Efficient low-redundancy codes for correcting multiple deletions," in Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms. SIAM, 2016, pp. 1884-1892.
[29] K. Cheng, Z. Jin, X. Li, and K. Wu, "Deterministic document exchange protocols, and almost optimal binary codes for edit errors," in 2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS). IEEE, 2018, pp. 200-211.
[30] J. Sima, N. Raviv, and J. Bruck, "Two deletion correcting codes from indicator vectors," IEEE Transactions on Information Theory, vol. 66, no. 4, pp. 2375-2391, 2020.
[31] J. Sima and J. Bruck, "Optimal k-deletion correcting codes," in 2019 IEEE International Symposium on Information Theory (ISIT), 2019, pp. 847-851.
[32] M. Abroshan, R. Venkataramanan, and A. G. i Fabregas, "Multilayer codes for synchronization from deletions," in 2017 IEEE Information Theory Workshop (ITW), Nov 2017, pp. 449-453.
[33] K. Tian, A. Fazeli, A. Vardy, and R. Liu, "Polar codes for channels with deletions," in 2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton), Oct 2017, pp. 572-579.
[34] G. Ganesan, "Correcting an ordered deletetion-erasure," in 2018 Wireless Advanced (WiAd). IEEE, 2018, pp. 1-5.
[35] ——, "Construction and redundancy of codes for correcting deletable errors," arXiv preprint arXiv:1805.00776, 2018.
[36] K. Tian, A. Fazeli, and A. Vardy, "Polar coding for deletion channels: Theory and implementation," in 2018 IEEE International Symposium on Information Theory (ISIT), 2018, pp. 1869-1873.
[37] I. Tal, H. D. Pfister, A. Fazeli, and A. Vardy, "Polar codes for the deletion channel: Weak and strong polarization," in 2019 IEEE International Symposium on Information Theory (ISIT). IEEE, 2019, pp. 1362-1366.
[38] K. Tian and R. Liu, "Scenario-simplified successive cancellation decoding of polar codes for channel with deletions," IEEE Access, vol. 7, pp. 18172-18182, 2019.
[39] M. C. Davey and D. J. MacKay, "Reliable communication over channels with insertions, deletions, and substitutions," IEEE Transactions on Information Theory, vol. 47, no. 2, pp. 687-698, 2001.
[40] E. A. Ratzer, "Marker codes for channels with insertions and deletions," in Annales des télécommunications, vol. 60, no. 1-2. Springer, 2005, pp. 29-44.
[41] V. Levenshtein, "Asymptotically optimum binary code with correction for losses of one or two adjacent bits," Problemy Kibernetiki, vol. 19, pp. 293-298, 1967.
[42] L. Cheng, T. G. Swart, H. C. Ferreira, and K. A. S. Abdel-Ghaffar, "Codes for correcting three or more adjacent deletions or insertions," in 2014 IEEE International Symposium on Information Theory, June 2014, pp. 1246-1250.
[43] C. Schoeny, A. Wachter-Zeh, R. Gabrys, and E. Yaakobi, "Codes correcting a burst of deletions or insertions," IEEE Transactions on Information Theory, vol. 63, no. 4, pp. 1971-1985, April 2017.
[44] P. P. Bours, "Codes for correcting insertion and deletion errors," 1994.
[45] G. M. Church, Y. Gao, and S. Kosuri, "Next-generation digital information storage in DNA," Science, vol. 337, no. 6102, pp. 1628-1628, 2012.
[46] -_, "Next-generation digital information storage in dna," Science, vol. 337, no. 6102, pp. 1628-1628, 2012.
[47] N. Goldman, P. Bertone, S. Chen, C. Dessimoz, E. M. LeProust, B. Sipos, and E. Birney, "Towards practical, high-capacity, low-maintenance information storage in synthesized dna," Nature, vol. 494, no. 7435, pp. 77-80, 2013.
[48] S. H. T. Yazdi, R. Gabrys, and O. Milenkovic, "Portable and error-free dna-based data storage," Scientific reports, vol. 7, no. 1, pp. 1-6, 2017.
[49] T. Batu, S. Kannan, S. Khanna, and A. McGregor, "Reconstructing strings from random traces," Departmental Papers (CIS), p. 173, 2004.
[50] T. Holenstein, M. Mitzenmacher, R. Panigrahy, and U. Wieder, "Trace reconstruction with constant deletion probability and related results." in SODA, vol. 8, 2008, pp. 389-398.
[51] A. McGregor, E. Price, and S. Vorotnikova, "Trace reconstruction revisited," in European Symposium on Algorithms. Springer, 2014, pp. 689-700.
[52] A. De, R. O'Donnell, and R. A. Servedio, "Optimal mean-based algorithms for trace reconstruction," in Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, 2017, pp. 1047-1056.
[53] 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, 2017, pp. 10421046.
[54] L. Hartung, N. Holden, and Y. Peres, "Trace reconstruction with varying deletion probabilities," in 2018 Proceedings of the Fifteenth Workshop on Analytic Algorithmics and Combinatorics (ANALCO). SIAM, 2018, pp. 54-61.
[55] Y. Peres and A. Zhai, "Average-case reconstruction for the deletion channel: subpolynomially many traces suffice," in 2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS). IEEE, 2017, pp. 228-239.
[56] N. Holden, R. Pemantle, and Y. Peres, "Subpolynomial trace reconstruction for random strings and arbitrary deletion probability," arXiv preprint arXiv:1801.04783, 2018.
[57] Z. Chase, "New lower bounds for trace reconstruction," arXiv preprint arXiv:1905.03031, 2019.
[58] M. Abroshan, R. Venkataramanan, L. Dolecek, and A. G. i FÃăbregas, "Coding for deletion channels with multiple traces," in 2019 IEEE International Symposium on Information Theory (ISIT), 2019, pp. 1372-1376.
[59] S. Kas Hanna and S. El Rouayheb, "Guess \& check codes for deletions and synchronization," in 2017 IEEE International Symposium on Information Theory (ISIT), June 2017, pp. 2693-2697.
[60] ——, "List decoding of deletions using guess \& check codes," in 2019 IEEE International Symposium on Information Theory (ISIT), 2019, pp. 2374-2378.
[61] _, "Guess \& check codes for deletions, insertions, and synchronization," IEEE Transactions on Information Theory, vol. 65, no. 1, pp. 3-15, Jan 2019.
[62] __, "Implementation of Guess \& Check (GC) codes," 2017, http://eceweb1.rutgers.edu/ csi/software.html.
[63] P. Elias, "List decoding for noisy channels," Quarterly Progress Report Massachusetts Institute of Technology, 1957.
[64] J. M. Wozencraft, "List decoding," Quarterly Progress Report, vol. 48, pp. 90-95, 1958.
[65] V. Guruswami and M. Sudan, "Improved decoding of reed-solomon and algebraic-geometric codes," in Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No.98CB36280), Nov 1998, pp. 28-37.
[66] R. Pellikaan and X.-W. Wu, "List decoding of q-ary reed-muller codes," IEEE Transactions on Information Theory, vol. 50, no. 4, pp. 679-682, April 2004.
[67] I. Tal and A. Vardy, "List decoding of polar codes," IEEE Transactions on Information Theory, vol. 61, no. 5, pp. 2213-2226, May 2015.
[68] A. Wachter-Zeh, "List decoding of insertions and deletions," IEEE Transactions on Information Theory, vol. 64, no. 9, pp. 6297-6304, Sept 2018.
[69] A. Orlitsky, "Interactive communication of balanced distributions and of correlated files," SIAM Journal on Discrete Mathematics, vol. 6, no. 4, pp. 548-564, 1993.
[70] S. Kas Hanna and S. El Rouayheb, "Correcting bursty and localized deletions using guess check codes," in 2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton), Oct 2017, pp. 9-16.
[71] ——, "Guess check codes for deletions and synchronization," in 2017 IEEE International Symposium on Information Theory (ISIT), June 2017, pp. 2693-2697.
[72] J. Sima and J. Bruck, "Optimal k-deletion correcting codes," in 2019 IEEE International Symposium on Information Theory (ISIT), 2019, pp. 847-851.
[73] W. Hoeffding, "Probability inequalities for sums of bounded random variables," Journal of the American Statistical Association, no. 58, pp. 13-30, 1963.

## Appendices

## Appendix A

## Additional Proofs for Chapter 2

## A. 1 Proof of Claim 2.6

Claim 2.6 is a generalization of Claim 2.5 for $\delta>1$ deletions. Recall that the decoder goes through $t$ cases where in each case corresponds to one possible way to distribute the $\delta$ deletions among the $k / \log k$ blocks. Claim 2.6 states that there exists a deterministic function $h$ of $\delta, h(\delta)$ independent of $k$, such that for any case $i, i=1,2, \ldots, t$,

$$
\operatorname{Pr}\left(\mathcal{Y}_{i}=\mathbf{Y} \mid \mathcal{P}_{1}=p_{1}, \ldots, \mathcal{P}_{\delta}=p_{\delta}\right) \leq \frac{h(\delta)}{q^{\frac{k}{\log k}-\delta}},
$$

where $\mathcal{Y}_{i}$ is the random variable representing the $q$-ary string decoded in case $i, i=1,2, \ldots, t$, and $\mathcal{P}_{r}, r=1, \ldots, c$, is the random variable representing the $r^{t h}$ MDS parity symbol.

To prove the claim, we follow the same approach used in the proof of Claim 2.5. Namely, we count the maximum number of different inputs (messages) that can generate the same output (decoded string) for $\delta$ fixed deletion positions $\left(d_{1}, \ldots, d_{\delta}\right)$ and $\delta$ given parities $\left(p_{1}, \ldots, p_{\delta}\right)$. Again, we call this number
$\gamma$. Recall that, for $\delta$ deletions, $\mathcal{Y}_{i}$ is decoded based on the first $\delta$ parities ${ }^{1}$. Hence, $\boldsymbol{\mathcal { Y }}_{i} \in \mathrm{~A}^{\delta}$, where

$$
\begin{align*}
& \mathrm{A}^{\delta} \triangleq \mathrm{A}_{1} \cap \mathrm{~A}_{2} \cap \ldots \cap \mathrm{~A}_{\delta}  \tag{A.1}\\
& \mathrm{A}_{\mathrm{r}} \triangleq\left\{\mathbf{Y} \in G F(q)^{k / \log k} \mid \mathbf{G}_{\mathbf{r}}^{\mathbf{T}} \mathbf{Y}=p_{r}\right\} \tag{A.2}
\end{align*}
$$

for $r=1, \ldots, \delta^{2}$. We are interested in showing that $\gamma$ is independent of the binary message length $k$. To this end, we upper bound $\gamma$ by a deterministic function of $\delta$ denoted by $h(\delta)$. Hence, we establish the following bound

$$
\begin{equation*}
\operatorname{Pr}\left(\mathcal{Y}_{i}=\mathbf{Y} \mid d_{1}, \ldots, d_{\delta}, p_{1}, \ldots, p_{\delta}\right) \leq \frac{\gamma}{\left|\mathrm{A}^{\delta}\right|} \leq \frac{h(\delta)}{q^{\frac{k}{\log k}-\delta}} \tag{A.3}
\end{equation*}
$$

We will now explain the approach for bounding $\gamma$ through an example for $\delta=2$ deletions.

Example A.1. Let $k=32$ and $\delta=2$. Consider the binary message $\mathbf{u}$ given by

$$
\mathbf{u}=b_{1} b_{2} \ldots b_{32}
$$

Its corresponding $q$-ary message $\mathbf{U}$ consists of 7 symbols (blocks) of length $\log k=5$ bits each. The message $\mathbf{u}$ is encoded into a codeword $\mathbf{x}$ using the GC code (Fig. 2.1). We assume that the first parity is the sum of the systematic symbols and the encoding vector for the second parity is (1, $\left.\alpha, \alpha^{2}, \alpha^{3}, \alpha^{4}, \alpha^{5}, \alpha^{6}\right)$ ${ }^{3}$. Moreover, we assume that the actual deleted bits in $\mathbf{x}$ are $b_{1}$ and $b_{7}$, but the chunking is done based on the assumption that deletions occurred in the $3^{\text {rd }}$ and $5^{\text {th }}$ block (wrong case). Similar to Example 2.5, it can be shown that the "free" bits are constrained by the following system of two linear equations in $G F(32)$,

$$
\left\{\begin{array}{c}
\alpha^{4} b_{1}+\alpha^{3} b_{7}+\alpha^{2} b_{13}+\alpha^{4} b_{14}+b_{15}+\alpha^{4} b_{16}+\alpha^{3} b_{22}+\alpha^{2} b_{23}+\alpha b_{24}+b_{25}=p_{1}^{\prime}  \tag{A.4}\\
\alpha^{4} b_{1}+\alpha^{4} b_{7}+\alpha^{2}\left(\alpha^{2} b_{13}+\alpha^{4} b_{14}+b_{15}\right)+\alpha^{7} b_{16}+\alpha^{5}\left(\alpha^{3} b_{22}+\alpha^{2} b_{23}+\alpha b_{24}+b_{25}\right)=p_{2}^{\prime}
\end{array}\right.
$$

To upper bound $\gamma$, we upper bound the number of solutions of the system given by (A.4). Equation (A.4)

[^19]can be written as follows
\[

\left\{$$
\begin{align*}
\left(b_{1}+b_{16}\right) \alpha^{4}+b_{7} \alpha^{3}+B_{1}+B_{2} & =p_{1}^{\prime}  \tag{A.5}\\
\left(b_{1}+b_{7}\right) \alpha^{4}+b_{16} \alpha^{7}+\alpha^{2} B_{1}+\alpha^{5} B_{2} & =p_{2}^{\prime}
\end{align*}
$$\right.
\]

where $B_{1}$ and $B_{2}$ are two symbols in $G F(32)$ given by

$$
\begin{align*}
& B_{1}=\alpha^{2} b_{13}+\alpha^{4} b_{14}+b_{15}  \tag{A.6}\\
& B_{2}=\alpha^{3} b_{22}+\alpha^{2} b_{23}+\alpha b_{24}+b_{25} \tag{A.7}
\end{align*}
$$

Notice that the coefficients of $B_{1}$ and $B_{2}$ in (A.5) originate from the MDS encoding vectors. Hence, for given bit values of $b_{1}, b_{7}$ and $b_{16}$, the MDS property implies that (A.5) has a unique solution for $B_{1}$ and $B_{2}$. Furthermore, since $B_{1}$ and $B_{2}$ have unique polynomial representations in $G F(32)$ of degree at most 4, for given values of $B_{1}$ and $B_{2}$, (A.6) and (A.7) have at most one solution for $b_{13}, b_{14}, b_{15}, b_{22}, b_{23}, b_{24}$ and $b_{25}$. We think of bits $b_{13}, b_{14}, b_{15}, b_{22}, b_{23}, b_{24}$ and $b_{25}$ as "free" bits of type $I$, and bits $b_{1}, b_{7}, b_{16}$ as "free" bits of type II. The previous reasoning indicates that for given values of the bits of type II, (A.4) has at most one solution. Therefore, an upper bound on $\gamma$ is given by the number of possible choices of the bits of type II. Hence, $\gamma \leq 2^{3}=8$.

Now, we generalize the previous example and upper bound $\gamma$ for $\delta>2$ deletions. Without loss of generality, we assume that the $\delta$ deletions occur in $\delta$ different blocks. Then, the "free" bits are constrained by a system of $\delta$ linear equations in $G F(q)$, where $q=k$. Let $\nu_{(.)}$and $\mu_{(.)}$be non-negative integers of value at most $k$. Each of the $\delta$ equations has the following form

$$
\begin{equation*}
\sum_{i=1}^{\beta} \alpha^{\nu_{i}} b_{\mu_{i}}+\sum_{j=1}^{\delta} \alpha^{\nu_{j}} B_{j}=p^{\prime} \tag{A.8}
\end{equation*}
$$

where $\beta$ is the number of "free" bits of type II, $B_{j} \in G F(q)$ is a linear combination of part of the bits in block $j$ ("free" bits of type I), and $p^{\prime} \in G F(q)$. The coefficients of $B_{j}, j=1, \ldots, \delta$, originate from the MDS code generator matrix. Hence, for given values of the bits of type II, the system of $\delta$ equations has a unique solution for $B_{j}, j=1, \ldots, \delta$. Furthermore, the linear combination of the bits that gives $B_{j}, j=1, \ldots, \delta$, has the following form

$$
\begin{equation*}
B_{j}=\alpha^{m} b_{j_{1}}+\alpha^{m-1} b_{j_{2}}+\ldots+\alpha^{m-\lambda_{j}+1} b_{j_{\lambda_{j}}} \tag{A.9}
\end{equation*}
$$

where $m<\log k$ is an integer, and $\lambda_{j}$ is the number of "free" bits of type I in $B_{j}$. Notice that (A.9) corresponds to a polynomial representation in $G F(q), q=k$, of degree less than $\log k$. Hence, for a given $B_{j}$, (A.9) has at most one solution. Therefore, $\gamma$ is upper bounded by the number of possible choices of the bits of type II, i.e., $\gamma \leq 2^{\beta}$. Recall that, when it is assumed that the block $j$ is affected by deletions, a sub-block of bits is chunked at the position of block $j$. However, because of the shift caused by the $\delta$ deletions, that sub-block may contain bits which do not originate from block $j$. The $\lambda_{j}$ "free" bits of type I in $B_{j}, b_{j_{1}}, \ldots, b_{j_{\lambda_{j}}}$, are the bits of the sub-block which do originate from block $j$. Since the shift at block $j$ is at most $\delta$ positions, it is easy to see that for any $j=1, \ldots, \delta$, we have $\lambda_{j} \geq \log k-\delta$. Therefore, since $\beta=\delta \log k-\sum_{j=1}^{\delta} \lambda_{j}$, we can use the lower bound on $\lambda_{j}$ to show that $\beta \leq \delta^{2}$. Hence, we obtain $\gamma \leq 2^{\delta^{2}} \triangleq h(\delta)^{4}$. In summary, we have shown that $\gamma$ is upper bounded by a deterministic function of $\delta$ that is independent of $k$.

Since the bound in (A.3) holds for arbitrary deletion positions $\left(d_{1}, \ldots, d_{\delta}\right)$, the upper bound on the probability of decoding failure in Theorem 2.1 holds for any $\delta$ deletion positions picked independently of the codeword. Moreover, for any given random distribution on the $\delta$ deletion positions (like the uniform distribution for example), we can apply the total law of probability and use the result from (A.3) to get

$$
\operatorname{Pr}\left(\mathcal{Y}_{i}=\mathbf{Y} \mid \mathcal{P}_{1}=p_{1}, \ldots, \mathcal{P}_{\delta}=p_{\delta}\right) \leq \frac{h(\delta)}{q^{\frac{k}{\log k}-\delta}}
$$

## A. 2 Proof of Theorem 2.7

We consider the same encoding scheme illustrated in Fig. 2.1 with the only modification that the message is chunked into blocks of length $\ell$ bits, and the field is size is $q=2^{\ell}$. Taking this modification into account, the proof of Theorem 2.7 follows the same steps of the proof of Theorem 2.1. It is easy to see from Fig. 2.1 that the redundancy here becomes $c(\delta+1) \ell$ bits. Also, the number of $q$-ary systematic symbols becomes $k / \ell$. Therefore, the total number of cases to be checked by the decoder is

$$
\begin{equation*}
t=\binom{k / \ell+\delta-1}{\delta}=\mathcal{O}\left(\frac{k^{\delta}}{\ell^{\delta}}\right) \tag{A.10}
\end{equation*}
$$

[^20]Furthermore, from the proof of Theorem 2.1 we have that the encoding complexity is

$$
\begin{equation*}
\mathcal{O}\left(c \cdot \frac{k}{\ell} \cdot \log ^{2} q\right)=\mathcal{O}(k \ell), \tag{A.11}
\end{equation*}
$$

and the decoding complexity is

$$
\begin{equation*}
\mathcal{O}\left(t \cdot \log ^{2} q \cdot k / \ell\right)=\mathcal{O}\left(\frac{k^{\delta+1}}{\ell^{\delta-1}}\right) \tag{A.12}
\end{equation*}
$$

As for the probability of decoding failure, the same intermediary steps of the proof of Theorem 2.1 apply for chunks of length $\ell$ instead of $\log k$. In particular, the key property used in the proofs of Claim 2.5 and Claim 2.6, is that each binary vector is mapped to a unique element in $G F(q)$. This property also applies here because the field size is $q=2^{\ell}$. Hence, from (2.20) we have

$$
\begin{equation*}
\operatorname{Pr}(F)<\frac{t \cdot h(\delta)}{q^{c-\delta}}=\mathcal{O}\left(\frac{(k / \ell)^{\delta}}{2^{\ell(c-\delta)}}\right) . \tag{A.13}
\end{equation*}
$$

From (A.13) we can see that the probability of decoding failure vanishes asymptotically if

$$
\begin{equation*}
\lim _{k \rightarrow+\infty} \frac{k^{\delta}}{\ell^{\delta} 2^{\ell(c-\delta)}}=0 \tag{A.14}
\end{equation*}
$$

which holds if $\ell=\Omega(\log k)$.

## Appendix B

## Additional Proofs for Chapter 3

## B. 1 Proof of Theorem 3.8 and Theorem 3.9

The proof of Theorems 3.8 and 3.9 is a direct generalization of the proof of Theorems 3.1 and 3.2. Next, we give the proof in terms of the chunking length $\ell$. Substituting $\ell=\log k$ and $\ell=w$ gives the results in Theorems 3.8 and 3.9, respectively. The redundancy follows from the construction (Section 3.5) where the $c \ell$ parity bits are repeated $(z w+1)$ times. Therefore, the redundancy is $c(z w+1) \ell$. The encoding complexity is $\mathcal{O}(k \ell)$, same as the case of a single window. The decoding complexity is given by the the product of the number of cases (guesses) and the complexity of decoding a constant number of systematic erasures (at most $2 z$ erasures). Hence, the decoding complexity is $\mathcal{O}\left(k^{z} \cdot k \ell\right)=\mathcal{O}\left(k^{z+1} \ell\right)$. As for the probability of decoding failure, the statement of Claim 3.12 can be generalized to $z>1$ windows as follows

$$
\begin{equation*}
\operatorname{Pr}\left(\mathcal{Y}_{i}=\mathbf{Y} \mid p_{1}, p_{2}, \ldots, p_{2 z}\right) \leq \frac{2^{z \ell}}{2^{k-2 z \ell}}=\frac{1}{2^{k-3 z \ell}} \tag{B.1}
\end{equation*}
$$

The previous statement follows from the same steps of the proof of Claim 3.12, when applied to $2 z$ equations, each corresponding to one MDS parity. Same as the proof of Theorems 3.1 and 3.2 , the rest
of the proof of follows from applying the union bound over the number of cases

$$
\begin{align*}
\operatorname{Pr}\left(F \mid p_{1}, \ldots, p_{c}\right) & \leq(t-1)\left|\mathrm{A}_{1}^{\mathrm{c}}\right| \operatorname{Pr}\left(\mathcal{Y}_{i}=\mathbf{Y} \mid p_{1}, \ldots, p_{2 z}\right)  \tag{B.2}\\
& \leq(t-1) \cdot 2^{k-c \ell} \cdot \frac{1}{2^{k-3 z \ell}}  \tag{B.3}\\
& =(t-1) 2^{-\ell(c-3 z)}  \tag{B.4}\\
& =\mathcal{O}\left(k^{z} 2^{-\ell(c-3 z)}\right) . \tag{B.5}
\end{align*}
$$

(B.2) follows from applying the union bound over the total number of cases $t$, similar to the proof of Theorems 3.1 and 3.2. (B.3) follows from (3.3) and (B.1). (B.5) follows from (3.2). Since the bound in (B.5) does not depend on the values of the parities, we obtain $\operatorname{Pr}(F)=\mathcal{O}\left(k^{z} 2^{-\ell(c-3 z)}\right)$.

## Appendix C

## Additional Proofs for Chapter 4

## C. 1 Proof of Lemma 4.6

We derive the bound in Lemma 4.6 based on the Chernoff-Hoeffding bound in [73].

Theorem C. 1 (Chernoff-Hoeffding bound [73]). Let $X \sim \operatorname{Binomial}(n, p)$, then for $k>n p$,

$$
\begin{equation*}
\operatorname{Pr}(X \geq k) \leq \exp (-n D(k / n \| p)) \tag{C.1}
\end{equation*}
$$

where $D(k / n \| p)$ is the Kullback-Leibler divergence between two Bernoulli random variables of parameters $k / n$ and $p$.

The number of deletions in a block $D$ follows a $\operatorname{Binomial}(n / k 2, k 1 / n)$ distribution. Therefore, by applying Theorem C.1, for $m>k_{1} / k_{2}$ we have

$$
\begin{equation*}
\operatorname{Pr}(X \geq m) \leq \exp \left[-\frac{n}{k_{2}} D\left(\frac{m k_{2}}{n} \| \frac{k_{1}}{n}\right)\right] . \tag{C.2}
\end{equation*}
$$

The Kullback-Leibler divergence is given by

$$
\begin{equation*}
D\left(\frac{m k_{2}}{n} \| \frac{k_{1}}{n}\right)=\frac{m k_{2}}{n} \ln \left(\frac{m k_{2}}{k_{1}}\right)+\left(\frac{n-m k_{2}}{n}\right) \ln \left(\frac{n-m k_{2}}{n-k_{1}}\right) . \tag{C.3}
\end{equation*}
$$

Therefore,

$$
\begin{equation*}
\operatorname{Pr}(X \geq m) \leq \exp \left[-\left(m \ln \left(\frac{m k_{2}}{k_{1}}\right)+\left(\frac{n-m k_{2}}{k_{2}}\right) \ln \left(\frac{n-m k_{2}}{n-k_{1}}\right)\right)\right] \tag{C.4}
\end{equation*}
$$

Let,

$$
\begin{equation*}
f(n) \triangleq\left(\frac{n-m k_{2}}{k_{2}}\right) \ln \left(\frac{n-m k_{2}}{n-k_{1}}\right) . \tag{C.5}
\end{equation*}
$$

The derivative of $f(n)$ with respect to $n$ is given by

$$
\begin{equation*}
f^{\prime}(n)=\frac{1}{k_{2}}\left[\ln \left(\frac{n-m k_{2}}{n-k_{1}}\right)+\frac{m k_{2}-k_{1}}{n-k_{1}}\right] . \tag{C.6}
\end{equation*}
$$

It is easy to verify that $f^{\prime}(n)<0$ for $m k_{2}>k_{1}$. Hence, $f(n)$ is decreasing function of $n$. Furthermore,

$$
\begin{align*}
\lim _{n \rightarrow+\infty} f(n) & =\lim _{n \rightarrow+\infty} \frac{\ln \left(\frac{n-m k_{2}}{n-k_{1}}\right)}{\frac{k_{2}}{n-m k_{2}}}  \tag{C.7}\\
& =\lim _{n \rightarrow+\infty} \frac{\frac{m k_{2}-k_{1}}{\left(n-k_{1}\right)\left(n-m k_{2}\right)}}{\frac{-k_{2}}{\left(n-m k_{2}\right)^{2}}}  \tag{C.8}\\
& =\frac{k_{1}-m k_{2}}{k_{2}} \lim _{n \rightarrow+\infty} \frac{n-m k_{2}}{n-k_{1}}  \tag{C.9}\\
& =\frac{k_{1}}{k_{2}}-m, \tag{C.10}
\end{align*}
$$

where (C.7) follows from applying L'Hospital's rule. Therefore, since $f(n)$ is a decreasing function, we have

$$
\begin{equation*}
f(n)>\frac{k_{1}}{k_{2}}-m \tag{C.11}
\end{equation*}
$$

Thus, by substituting in (C.4) we get

$$
\begin{equation*}
\operatorname{Pr}(X \geq m)<\exp \left[-\left(m \ln \left(\frac{m k_{2}}{k_{1}}\right)+\frac{k_{1}}{k_{2}}-m\right)\right]=\left(\frac{e k_{1}}{m k_{2}}\right)^{m} e^{-\frac{k_{1}}{k_{2}}} \tag{C.12}
\end{equation*}
$$


[^0]:    3.1. Summary of the notations used in the chapter.57

[^1]:    ${ }^{1}$ The non-linearity implies that these codes cannot be made systematic by a linear transformation.

[^2]:    ${ }^{2}$ In this example, we use the field $G F(17)$ to simplify the calculations. In Chapter 2 , we use the extension field $G F\left(2^{4}\right)$ for blocks of size $\log k=4$ bits.

[^3]:    ${ }^{3} \mathrm{VT}$ codes are still used for segments affected by only one deletion.

[^4]:    The results described in this chapter were published in part in the IEEE International Symposium on Information Theory (ISIT) 2017 [59], IEEE International Symposium on Information Theory (ISIT) 2019 [60], and also in the IEEE Transactions on Information Theory [61].
    ${ }^{1}$ A decoding failure event corresponds to the decoder not being able to decode the transmitted sequence and outputting "failure to decode" error message. When the decoder does not output the error message, the decoded string is guaranteed to be the correct one.
    ${ }^{2}$ Codes that can correct $\delta$ oblivious deletions can also correct $\delta$ deletions that are randomly distributed based on any particular distribution.

[^5]:    ${ }^{3}$ Explicit constructions of systematic Reed-Solomon codes (based on Cauchy or Vandermonde matrices) always exist for these parameters.

[^6]:    ${ }^{4}$ VT codes can correct one deletion with zero-error. However, GC codes are generalizable to multiple deletions.

[^7]:    ${ }^{5}$ After chunking, the last block may contain fewer than $\log k$ bits. In order to map the block to its corresponding symbol, it is first padded with zeros to a length of $\log k$ bits. Hence, $\mathbf{U}$ consists of $\lceil k / \log k\rceil$ symbols. We drop the ceiling notation throughout the chapter and simply write $k / \log k$.

[^8]:    ${ }^{6}$ The complexity of checking whether a decoded erasure, of length $\log k$ bits, is a supersequence of its corresponding sub-block is $\mathcal{O}\left(\log ^{2} k\right)$ using the Wagner-Fischer algorithm. Hence, Criterion 2 (Definition 2.4) does not affect the order of decoding complexity.

[^9]:    ${ }^{7}$ For $\ell=k$ and $c=1$, the code is equivalent to a $(\delta+1)$ repetition code.

[^10]:    ${ }^{8}$ These results are for $\ell=\log k$, the decoding time can be decreased if we increase $\ell$ (trade-off A, Section 2.7).
    ${ }^{9}$ VT codes are still used for segments affected by only one deletion.

[^11]:    ${ }^{10}$ Polynomial time in terms of the length of the codeword $n$.

[^12]:    ${ }^{11}$ Depending on the runs of bits within the received string $\mathbf{y}$, different guesses may lead to the same decoded string.

[^13]:    The results described in this chapter were published in part in the 2017 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton) [70], and also submitted to the IEEE Transactions on Information Theory.
    ${ }^{1}$ An implementation of GC codes can be found and tested online on the link in [62].

[^14]:    ${ }^{2}$ The term decoding failure means that the decoder cannot make a correct decision and outputs a "failure to decode" error message.

[^15]:    ${ }^{3}$ The MDS generator matrix used in this example is based on a Vandermonde matrix. For the general case we construct the generator matrix by concatenating an identity matrix with a Cauchy matrix.

[^16]:    ${ }^{4}$ The information is in the systematic bits, hence the decoder does need to recover parity bits.

[^17]:    ${ }^{5}$ The decoder knows the values of $k, n$, and $w$ (code parameters), and can determine the value of $\delta$ by calculating the difference between $n$ and the length of the received string.

[^18]:    ${ }^{6}$ The window of $w$ bits, in which the $\delta$ deletions are localized, is fixed.
    ${ }^{7}$ The last symbol is padded to a length of 5 bits by adding zeros.
    ${ }^{8}$ The extension field used is $G F(32)$ and has a primitive element $\alpha$, with $\alpha^{5}=\alpha^{2}+1$.

[^19]:    ${ }^{1}$ The underlying assumption here is that the $\delta$ deletions affected exactly $\delta$ blocks. In cases where it is assumed that less than $\delta$ blocks are affected, then less than $\delta$ parities will be used to decode $\mathcal{Y}_{\boldsymbol{i}}$, and the same analysis applies.
    ${ }^{2}$ The set $\mathrm{A}^{\delta}$ is the generalization of set $\mathrm{A}_{1}$ for $\delta=1$.
    ${ }^{3}$ The extension field used is $G F(32)$ and has a primitive element $\alpha$, with $\alpha^{5}=\alpha^{2}+1$.

[^20]:    ${ }^{4}$ If the $\delta$ deletions occur in $z<\delta$ blocks, then $\beta=z \log k-\sum_{j=1}^{z} \lambda_{j}$, and the upper bound $\beta \leq \delta^{2}$ would still hold.

