Coding for Synchronization Errors

Amirbehshad Shahrasbi

CMU-CS-20-131

September 2020

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Bernhard Haeupler (Chair)
Venkatesan Guruswami
Rashmi Vinayak
Mahdu Sudan (Harvard University)
Sergey Yekhanin (Microsoft Research)

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

This research was sponsored by the National Science Foundation under grant numbers CCF-1527110, CCF-1618280, CCF-1750808, CCF-1814603, and CCF-1910588 and the Alfred P. Sloan Foundation under grant number FG201911557. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: Synchronization, Error-Correction for Synchronization Errors, Coding for Insertions and Deletions, Synchronization Strings, List Decoding for Insertions and Deletions, Interactive Coding for Synchronization Errors, Edit Distance Computation

To my parents and my sister

Abstract

Coding theory is the study of algorithms and techniques that facilitate reliable information transmission over noisy mediums, most notably through combinatorial objects called error-correcting codes. Following the inspiring works of Shannon and Hamming, a sophisticated and extensive body of research on error-correcting codes has led to a deep and detailed theoretical understanding as well as practical implementations that have helped fuel the digital revolution. Error-correcting codes can be found in essentially all modern communication, computation, and data storage systems. While being remarkably successful in understanding the theoretical limits and trade-offs of reliable communication under errors and erasures, the coding theory literature significantly lags behind when it comes to overcoming errors that concern the timing of communications. In particular, the study of correcting synchronization errors, i.e., symbol insertions and deletions, while initially introduced by Levenshtein in the 60s, has significantly fallen behind our highly sophisticated knowledge of codes for Hamming-type errors.

This thesis investigates coding against synchronization errors under a variety of models and attempts to understand trade-offs between different qualities of interest in respective coding schemes such as rate, distance, and algorithmic qualities of the code. Most of the presented results rely on synchronization strings, simple yet powerful pseudorandom objects introduced in this work that have proven to be very effective solutions for coping with synchronization errors in various settings.

Through indexing with strings that satisfy certain pseudo-random properties, we provide synchronization codes that achieve near-optimal rate-distance trade-off. We further attempt to provide constructions that enable fast encoding/decoding procedures. We study the same problem under the list-decoding regime, where the decoder is expected to provide a short list of codewords that is guaranteed to contain the sent message. We will also try to better understand the fundamental limits of list-decoding for synchronization errors such as the list-decoding capacity or maximal error resilience for list-decodable synchronization codes. This thesis furthermore studies synchronization strings and other related pseudo-random string properties as combinatorial objects that are of independent interest. Such combinatorial objects will be used to extend some of our techniques to alternative communication problems such as coding from block transposition errors or coding for interactive communication.

Acknowledgments

I would like to start this section by expressing my great gratitude towards my advisor, Bernhard Haeupler for his kind support and encouragement throughout my years at CMU. Bernhard is a great mentor with amazing teaching abilities. I was very fortunate to work under his supervision and learned a lot from his outstanding technical knowledge and intuition.

I would also like to thank many amazing researchers that I had the pleasure of collaborating with: Madhu Sudan, Venkatesan Guruswami, Xin Li, Aviad Rubinstein, Ellen Vitercik, Kuan Cheng, and Ke Wu. I thank Sergey Yekhanin for an enjoyable and productive internship at Microsoft Research at Redmond, WA during the summer of 2019. I also appreciate the time that members of my thesis committee spent on reviewing this dissertation and providing valuable feedback.

On a more personal note, I want to take a moment to thank many amazing friends that I met in Pittsburgh. The fun and memorable times that I spent with them helped me greatly through the challenging years of my Ph.D. studies and made Pittsburgh feel like home to me. I also thank the friends from home with whom I stayed in touch and had occasional yet very uplifting and energizing video calls.

And last, but certainly not the least, I would like to thank my family: my mom, my dad, and my sister Yasaman. Due to certain aggressively restrictive U.S. visa policies, my Ph.D. studies at CMU came at the very hefty cost of not being able to see them for five painfully long years. I thank them from the bottom of my heart for their support and encouragement and humbly dedicate this dissertation to them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Following the inspiring works of Shannon and Hamming, the field of coding theory has advanced our understanding of how to efficiently correct symbol substitution and erasure errors occurring during a communication. The practical and theoretical impact of error-correcting codes on technology and engineering as well as mathematics, theoretical computer science, and other fields is hard to overestimate. The sophisticated and extensive body of research on error-correcting codes has led to a deep and detailed theoretical understanding as well as practical implementations that have helped fuel the Digital Revolution. Error-correcting codes can be found in essentially all modern communication and computation systems.

While being remarkably successful in understanding the theoretical limits and trade-offs of reliable communication under symbol substitution and erasure errors, the coding theory literature lags significantly behind when it comes to overcoming errors that concern the timing of communications. In particular, the study of correcting synchronization errors, i.e., symbol insertions and deletions, while initially introduced by Levenshtein in the 60s, has significantly fallen behind our highly sophisticated knowledge of codes for Hamming-type errors.

This discrepancy has been well noted in the literature. An expert panel [GDR+63] in 1963 concluded: *"There has been one glaring hole in [Shannon's] theory; viz., uncertainties in timing, which I will propose to call time noise, have not been encompassed .... Our thesis here today is that the synchronization problem is not a mere engineering detail, but a fundamental communication problem as basic as detection itself!"* however as noted in a comprehensive survey [MBT10] in 2010: *"Unfortunately, although it has early and often been conjectured that error-correcting codes capable of correcting timing errors could improve the overall performance of communication systems, they are quite challenging to design, which partly explains why a large collection of synchronization techniques not based on coding were developed and implemented over the years."* or as Mitzenmacher puts in his survey [Mit09]: *"Channels with synchronization errors, including both insertions and deletions as well as more general timing errors, are simply not adequately understood by current theory. Given the near-complete knowledge we have for channels with erasures and errors ... our lack of understanding about channels with synchronization errors is truly remarkable."*

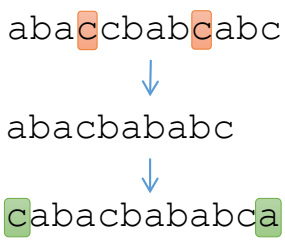


Figure 1.1: An example depicting insertion and deletion errors. Two deletions in the highlighted positions from the top string would convert it into the middle one and two insertions in the highlighted positions into the middle string would result into the bottom string.

We, too, believe that the current lack of good codes and general understanding of how to handle synchronization errors is the reason why systems today still spend significant resources and efforts on keeping very tight controls on synchronization while other noise types are handled more efficiently using coding techniques. We are convinced that a better theoretical understanding together with practical code constructions will eventually lead to systems that naturally and more efficiently use coding techniques to address synchronization and noise issues jointly. There are already several emerging application areas, such as DNA-storage [OAC+17, BGH+16, GBC+13, CGK12, YKGR+15, BLC+16] that significantly highlight the need for high-quality synchronization coding schemes. Besides, we feel that better understanding the combinatorial structure underlying (codes for) insertions and deletions will have an impact on other parts of mathematics and theoretical computer science. This thesis is dedicated to studying and better understanding codes for synchronization errors.

1.1 Synchronization Errors

In this thesis, we are concerned with the reliable communication of streams of symbols through a given noisy channel. There are two basic types of noise that we will consider, *Hamming-type errors* and *synchronization errors*. Hamming-type errors consist of *erasures*, that is, a symbol being replaced with a special "?" symbol indicating the erasure, and *substitutions* where a symbol is replaced with another symbol of the alphabet. Synchronization errors consist of *deletions*, that is, a symbol being removed without replacement, and *insertions*, where a new symbol is added somewhere within the stream. (see Fig. 1.1 for an example)

Synchronization errors are strictly more general and harsher than Hamming-type errors. In particular, any symbol substitution can also be achieved via a deletion followed by an insertion at the same location and any erasure can be interpreted as a deletion together with the extra information of where this deletion has taken place. This shows that any error pattern generated by k Hamming-type errors can also be replicated using k synchronization errors[1], making dealing with synchronization errors at least as hard as Hamming-type

[1]See Chapter 3 for a precise argument using a proper measure to count Hamming-type errors.

errors. The real problem that synchronization errors bring with them, however, is that they cause sending and receiving parties to become "out of sync". This easily changes how received symbols are interpreted and makes designing codes or other systems tolerant to synchronization errors an inherently more difficult and significantly less understood problem.

1.2 Scope of the Thesis

The study of coding for synchronization errors was initiated by Levenshtein [Lev65] in 1966 when he showed that Varshamov-Tenengolts codes can correct a single insertion, deletion, or substitution error with an optimal redundancy of almost $\log n$ bits. Ever since, synchronization errors have been studied in various settings. In this section, we categorize some of the commonly studied settings and specify the one relevant to this thesis.

The first important aspect is the noise model. Several works have studied coding for synchronization errors under the assumption of random errors, most notably, to study the capacity of deletion channels which independently delete each of the transmitting symbols with some fixed probability. (See [Mit09, MBT10, CR20].) This thesis exclusively focuses on worst-case error models in which correction has to be possible from any (adversarial) error pattern bounded only by the total number of insertions and deletions.

Another angel to categorize the previous work on codes for synchronization error is the noise regime. In the same spirit as ordinary error-correcting codes, the study of families of synchronization codes have included both ones that protect against a fixed number of synchronization errors and ones that consider error count that is a fixed fraction of the block length. The inspiring work of Levenshtein [Lev65] falls under the first category and is followed by several works designing synchronization codes correcting k errors for specific values of k [Slo02, Ten84, HF02, GS18] or with k as a general parameter [AGPFC11, BGZ18]. In this thesis, we focus on the second category, i.e., infinite families of synchronization codes with increasing block length that are defined over a fixed alphabet size and can correct from constant-fractions of worst-case synchronization errors.

We, furthermore, mainly focus on codes that can be efficiently constructed and decoded – in contrast to merely existential results. The first such code was constructed in 1999 by Schulman and Zuckerman [SZ99]. They provided an efficient, asymptotically good synchronization code with a constant rate and a constant distance. We will further discuss the previous work and how contributions of this thesis fit into them when describing the contributions in the following chapters.

1.3 Basics of Error Correcting Codes and InsDel Codes

In this section, we give a general and informal description of error correcting codes (ECCs) and insertion-deletion codes (InsDel codes) along with some basic notions associated with them. We defer a more formal introduction to Chapter 2.

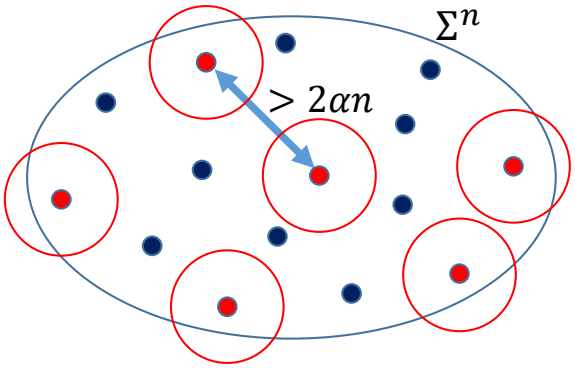


Figure 1.2: In a code with a minimum distance of more than $2\alpha n$, balls of radius αn around codewords are disjoint. Hence, unique decoding from αn substitutions is possible.

1.3.1 Error Correcting Codes

Consider the scenario where two parties, Alice and Bob, are communicating over a noisy communication channel. More precisely, we assume that Alice can use this channel n times to send over a symbol out of an alphabet Σ to Bob. Let us denote these symbols as $x_1, x_2, \cdots, x_n$. We assume that the *noise* is modeled by an adversary that can apply αn errors in the form of substituting one of the transmitted symbols to another symbol of the alphabet Σ for some $\alpha \in (0, 1)$. Alice and Bob aim to conduct a reliable communication over such a channel, i.e., communicate in a way that Bob is able to recover the message sent by Alice and yet be able to convey as much "information" as possible.

Error-correcting codes are natural solutions for the scenario set forth above. Qualitatively speaking, an ECC is a set of strings (or *codewords*) in Σ^n that are "very different" from each other – so much that Bob would be able to uniquely identify which codeword it was that Alice has sent.

More precisely, we define the *Hamming distance* between two strings $s_1, s_2 \in \Sigma^n$ as the number of positions $i \in \{1, 2, \cdots, n\}$ where s_1 and s_2 do not agree, i.e., $s_1[i] \neq s_2[i]$. An error-correcting code $C \in \Sigma^n$ is said to have a minimum distance d if there is a Hamming distance of d or more between any pair of codewords $w_1, w_2 \in C$. Note that if Alice exclusively sends the codewords of the code C and the minimum distance of C is larger than $2n\alpha$, then Bob will be able to uniquely recover the string sent by Alice. One can verify this by thinking of αn radius balls around each codeword w (that is the set of all strings with Hamming distance αn or less to w). Note that if Alice sends a codeword w, what Bob will receive would reside inside this ball. If the minimum distance of the code is larger than $2\alpha n$, then these balls are disjoint and, therefore, Bob is able to identify the string sent by Alice using its distorted version.

Note that an interesting trade-off for ECCs is that "For a given n, Σ, and minimum distance d, what is the largest possible size for a code $C \in \Sigma^n$ with minimum distance d?". Note that using the code C, Alice can essentially send a number between one and $|C|$ to Bob. This is equivalent to $\log |C|$ bits of information.[2] The notion of *rate* is defined as

[2]Throughout this document, all logarithms are binary unless stated otherwise.

the amount of information that Alice can convey using code $|C|$ normalized by the total amount of information that could be conveyed over the channel in the absence of noise, i.e., $n \log |\Sigma|$. A natural goal in the design of error-correcting codes is to find ones that attain as high of a rate as possible.

There are also several interesting computational angles in the study of EECs. The encoding procedure of an ECC C is the algorithm that Alice uses to compute the codewords of C, i.e., one that computes (some) injective function from $\{1, 2, \cdots, |C|\}$ to C. Further, the decoding procedure of C would be the algorithm that Bob uses to derive the codeword sent by Alice using the distorted version of it. Finding codes with fast encoding and decoding procedures has been an interesting algorithmic challenge in the study of ECCs.

1.3.2 Insertion-Deletion Codes

Insertion-deletion codes are equivalents of ECCs but for channels where the adversary is capable of performing symbol insertion or symbol deletion errors. To define insdel codes formally, we start with the definition of the *edit distance* which is the equivalent of the notion of Hamming distance for ECCs. The *edit distance* between two strings is defined as the smallest number of insertions and deletions required to turn one into the other one.

In the same spirit as ECCs, insdel codes are defined as sets of strings that have a "large" minimum edit-distance. More precisely, and insdel code $C \in \Sigma^n$ with minimum edit-distance $2\alpha n$, would enable unique-decoding from αn synchronization errors on the receiver's end. The notions of rate, encoding complexity, and decoding complexity are defined in the same manner as in ECCs.

1.4 Thesis Contributions and Structure

We begin the rest of this thesis by setting notations and presenting some preliminary definitions in Chapter 2. Following is a brief description of the problems studied in each of the chapters of this thesis.

Chapter 3: Coding via Indexing and Approaching the Singleton Bound

In Chapter 3, we introduce *synchronization strings*, which provide a novel way to efficiently deal with synchronization errors. For every $\varepsilon > 0$, synchronization strings allow to index a sequence with an $\varepsilon^{-O(1)}$ size alphabet such that one can efficiently transform k synchronization errors into $(1 + \varepsilon)k$ Hamming-type errors.

A straightforward application of our synchronization strings based indexing method gives a simple black-box construction which transforms any error-correcting code (ECC) into an equally efficient insertion-deletion (insdel) code with only a small increase in the alphabet size. This instantly transfers much of the highly developed understanding for regular ECCs into the realm of insdel codes. Most notably, for the complete noise spectrum we obtain efficient "near-MDS" insdel codes which get arbitrarily close to the optimal rate-distance tradeoff given by the Singleton bound. In particular, for any $\delta \in (0, 1)$ and $\varepsilon > 0$,

we give a family of insdel codes achieving a rate of $1-\delta-\varepsilon$ over a constant size alphabet that efficiently corrects a δ fraction of insertions or deletions.

Chapter 4: List-decoding over Large Alphabets

In Chapter 4, we study codes that are list-decodable under insertions and deletions. Specifically, we consider the setting where, given a codeword x of length n over some finite alphabet Σ of size q, $\delta \cdot n$ codeword symbols may be adversarially deleted and $\gamma \cdot n$ symbols may be adversarially inserted to yield a corrupted word w. A code is said to be list-decodable if there is an (efficient) algorithm that, given w, reports a small list of codewords that include the original codeword x. Given δ and γ we study what is the rate R for which there exists a constant q and list size L such that there exist codes of rate R correcting δ-fraction insertions and γ-fraction deletions while reporting lists of size at most L.

Using the concept of synchronization strings, we show some surprising results. We show that for every $0 \leq \delta < 1$, every $0 \leq \gamma < \infty$ and every $\varepsilon > 0$ there exist codes of rate $1-\delta-\varepsilon$ and constant alphabet (so $q = O_{\delta,\gamma,\varepsilon}(1)$) and sub-logarithmic list sizes. Furthermore, our codes are accompanied by efficient (polynomial time) decoding algorithms. We stress that the fraction of insertions can be arbitrarily large (more than 100%), and the rate is independent of this parameter. We also prove several tight bounds on the parameters of list-decodable insdel codes. In particular, we show that the alphabet size of insdel codes needs to be exponentially large in ε^{-1}, where ε is the gap to capacity above.

These results shed light on the remarkable asymmetry between the impact of insertions and deletions from the point of view of error-correction: Whereas deletions cost in the rate of the code, insertion costs are borne by the adversary and not the code! Our results also highlight the dominance of the model of insertions and deletions over the Hamming model: A Hamming error is equal to one insertion and one deletion (at the same location). Thus the effect of δ-fraction Hamming errors can be simulated by δ-fraction of deletions and δ-fraction of insertions — but insdel codes can deal with much more insertions without loss in rate (though at the price of higher alphabet size).

Chapter 5: Optimally Resilient Codes for List-Decoding

In Chapter 5, we give a complete answer to the following basic question: "What is the maximal fraction of deletions or insertions tolerable by q-ary list-decodable codes with non-vanishing information rate?". This question has been open even for binary codes, including the restriction to the binary insertion-only setting, where the best-known result was that a $\gamma \leq 0.707$ fraction of insertions is tolerable by some binary code family.

For any desired $\varepsilon > 0$, we construct a family of binary codes of positive rate which can be efficiently list-decoded from any combination of γ fraction of insertions and δ fraction of deletions as long as $\gamma + 2\delta \leq 1 - \varepsilon$. On the other hand, for any γ, δ with $\gamma + 2\delta = 1$ list-decoding is impossible. Our result thus precisely characterizes the feasibility region of binary list-decodable codes for insertions and deletions.

We further generalize our result to codes over any finite alphabet of size q. Surprisingly, our work reveals that the feasibility region for $q > 2$ is *not* the natural generalization of the binary bound above. We provide tight upper and lower bounds that precisely pin down

the feasibility region, which turns out to have a $(q-1)$-piece-wise linear boundary whose q corner-points lie on a quadratic curve.

Chapter 6: Rate vs. Distance for List-Decoding

In this chapter, we provide results that further complete the picture portrayed by the results of Chapters 4 and 5 in regard to list-decodable insdel codes. We prove several bounds on the list-decoding capacity of worst-case synchronization channels, i.e., the highest rate that is achievable for q-ary list-decodable insdel codes that can correct from δ fraction of deletions and γ fraction of insertions. We present upper-bounds and lower-bounds for the capacity for the cases of insertion-only channels ($\delta = 0$), deletion-only channels ($\gamma = 0$), and the generalized case of channels with both insertions and deletions. Our lower-bounds are derived by analysis of random codes.

Note that this question generalizes the questions that Chapters 4 and 5 answer. Chapter 4 finds the maximal achievable rate for codes that can correct (γ, δ) fraction of insdel errors while allowing the alphabet size to be sufficiently large (therefore, ignoring the alphabet size) and Chapter 5 pins down the resilience region, i.e., the set of (γ, δ) error fractions for which capacity is non-zero. These are both special cases of the question of interest in this chapter.

The results of this chapter also give interesting implications on the code constructions from Chapters 3 and 4.We show that the alphabet size of insdel codes needs to be exponentially large in ε^{-1}, where ε is the gap to capacity above. Our result even applies to settings where the unique-decoding capacity equals the list-decoding capacity and when it does so, it shows that the alphabet size needs to be exponentially large in the gap to capacity. This is sharp contrast to the Hamming error model where alphabet size polynomial in ε^{-1} suffices for unique decoding. This lower bound also shows that the exponential dependence on the alphabet size in Chapter 3 is actually necessary!

Chapter 7: Online Repositioning, Channel Simulation and Interactive Coding

In Chapter 7, we use an online repositioning algorithm for synchronization strings to present many new results related to reliable (interactive) communication over insertion-deletion channels.

We show how to hide the complications of synchronization errors in many applications by introducing very general *channel simulations* which efficiently transform an insertion-deletion channel into a regular symbol substitution channel with an error rate larger by a constant factor and a slightly smaller alphabet. Our channel simulations depend on the fact that, at the cost of increasing the error rate by a constant factor, synchronization strings can be decoded in a streaming manner that preserves linearity of time. Interestingly, we provide a lower bound showing that this constant factor cannot be improved to $1 + \varepsilon$, in contrast to what is achievable for error correcting codes. These channel simulations drastically and cleanly generalize the applicability of synchronization strings.

Using such channel simulations, we provide interactive coding schemes which simulate any interactive two-party protocol over an insertion-deletion channel. These results improve over the state-of-the-art interactive coding schemes of Braverman et al.[BGMO17]

and Sherstov and Wu [SW19] which achieve a small constant rate and require exponential time computations with respect to computational and communication complexities. We provide the first computationally efficient interactive coding scheme for synchronization errors, the first coding scheme with a rate approaching one for small noise rates, and also the first coding scheme that works over arbitrarily small alphabet sizes.

Finally, using our channel simulations, we provide an explicit binary insertion-deletion code achieving a rate of $1 - O(\sqrt{\delta \log(1/\delta)})$ that improves over the codes by Guruswami and Wang [GW17] in terms of the rate-distance trade-off. The codes of Guruswami and Wang [GW17] were the state-of-the-art codes at the time of publishing this result. As we will mention in Chapter 7, further improvements have been made in this regard ever since.

Chapter 8: Local Repositioning, Near-Linear Time Decoding

In Chapter 8, we present several algorithmic results for synchronization strings:

- We give a deterministic, linear time synchronization string construction, improving over an $O(n^5)$ time randomized construction in Chapter 4.
- We give a deterministic construction of an infinite synchronization string which outputs the first n symbols in $O(n)$ time.
- Both synchronization string constructions are *highly explicit*, i.e., the i^{th} symbol can be deterministically computed in $O(\log i)$ time.
- This chapter also introduces a generalized notion called *long-distance synchronization strings*. Such strings allow for *local and very fast* decoding. In particular only $O(\log^3 n)$ time and access to logarithmically many symbols is required to reposition any index.

This chapter also provides several applications for these improved synchronization strings:

- For any $\delta < 1$ and $\varepsilon > 0$, we provide an insertion-deletion code with rate $1 - \delta - \varepsilon$ which can correct any $\delta/3$ fraction of insertion and deletion errors in $O(n \log^3 n)$ time. This near linear computational efficiency is surprising given that we do not even know how to compute the (edit) distance between the decoding input and output in sub-quadratic time.
- We show that local decodability implies that error correcting codes constructed with long-distance synchronization strings can not only efficiently recover from δ fraction of insdel errors but, similar to [SZ99], also from any $O(\delta/\log n)$ fraction of *block transpositions and block replications*. These block corruptions allow arbitrarily long substrings to be swapped or replicated anywhere.
- We show that highly explicitness and local decoding allow for *infinite channel simulations with exponentially smaller memory and decoding time requirements*. These simulations can then be used to give the first *near-linear time interactive coding scheme for insertions and deletions*, similar to the result of [BN13] for Hamming errors.

Chapter 9: Approximating Edit Distance via Indexing, Near-Linear Time Decoding

In Chapter 9, we introduce *fast-decodable indexing schemes for edit distance* which can be used to speed up edit distance computations to near-linear time if one of the strings is indexed by an indexing string I. In particular, for every length n and every $\varepsilon > 0$, one can, in near-linear time, construct a string $I \in \Sigma'^n$ with $|\Sigma'| = O_\varepsilon(1)$, such that, indexing any string $S \in \Sigma^n$ with I (i.e., concatenating S symbol-by-symbol with I) results in a string $S' \in \Sigma''^n$ where $\Sigma'' = \Sigma \times \Sigma'$ for which edit distance computations are easy, i.e., one can compute a $(1+\varepsilon)$-approximation of the edit distance between S' and any other string in $O(n\mathsf{poly}(\log n))$ time.

Our indexing schemes can be used to improve the decoding complexity of the state-of-the-art error correcting codes for insertions and deletions. In particular, they lead to near-linear time decoding algorithms for the insertion-deletion codes from Chapter 3 and faster decoding algorithms for list-decodable insertion-deletion codes from Chapter 4. Interestingly, the latter codes are a crucial ingredient in the construction of fast-decodable indexing schemes.

Chapter 10: Combinatorial Properties of Synchronization Strings

In Chapter 10, we study combinatorial properties of synchronization strings.

In Chapter 3, we show that for any parameter $\varepsilon > 0$, synchronization strings of arbitrary length exist over an alphabet whose size depends only on ε. Specifically, we obtain an alphabet size of $O(\varepsilon^{-4})$, which leaves an open question on where the minimal size of such alphabets lies between $\Omega(\varepsilon^{-1})$ and $O(\varepsilon^{-4})$. In this chapter, we partially bridge this gap by providing an improved lower bound of $\Omega\left(\varepsilon^{-3/2}\right)$, and an improved upper bound of $O\left(\varepsilon^{-2}\right)$. We also provide fast explicit constructions of synchronization strings over small alphabets.

Further, along the lines of previous work on similar combinatorial objects, we study the extremal question of the smallest possible alphabet size over which synchronization strings can exist for some constant $\varepsilon < 1$. We show that one can construct ε-synchronization strings over alphabets of size four while no such string exists over binary alphabets. This reduces the extremal question to whether synchronization strings exist over ternary alphabets.

1.5 Bibliographical Remarks

This thesis is based on collaborations of the author with several brilliant researchers. The results presented in Chapter 3 are based on a joint work with Bernhard Haeupler [HS17]. Chapters 4 and 6 are based on joint works with Bernhard Haeupler and Madhu Sudan [HSS18, HS20]. Chapter 5 states our findings in a collaboration with Venkatesan Guruswami and Bernhard Haeupler [GHS20]. Chapter 7 is based on a joint work with Bernhard Haeupler and Ellen Vitercik [HSV18]. Chapter 8 presents the results of [HS18]. A collaboration with Aviad Rubinstein and Bernhard Haeupler [HRS19] lead to the results discussed in Chapter 9 and, finally, Chapter 10 is based on a joint work with Xin Li, Bernhard Haeupler, Kuang Chen, and Ke Wu [CHL+19].

Chapter 2

Preliminaries and Notation

2.1 Notations

In this section, we set some notation that will be frequently used throughout this thesis.

We often use $[n]$ to denote the set $\{1, 2, \cdots, n\}$. For two strings $S \in \Sigma^n$ and $S' \in \Sigma^{n'}$ over alphabet Σ, we denote their concatenation with $S \cdot S' \in \Sigma^{n+n'}$. For any positive integer k, S^k is defined as k copies of S concatenated together and for integers $1 \leq i \leq j \leq n$, we denote the substring of S starting from the ith index through and including the jth one by $S[i, j]$. Such a substring is also called a *factor* of S. Further, for $i < 1$ we define $S[i, j] = \perp^{-i+1} \cdot S[1, j]$ where $\perp$ is a special symbol not included in Σ. We denote the substring from the ith index through, but not including, the jth index by $S[i, j)$. Substrings $S(i, j]$ and $S(i, j)$ are similarly defined. Finally, $S[i]$ denotes the ith symbol of string S and $|S|$ denotes the length of S. Occasionally, the alphabets we use are the Cartesian product of several alphabets, i.e. $\Sigma = \Sigma_1 \times \cdots \times \Sigma_n$. If T is a string over Σ, then we write $T[i] = (a_1, \ldots, a_n)$ where $a_i \in \Sigma_i$. We often use the notion of *indexing* defined as follows.

Definition 2.1.1 (String Indexing/Symbol-Wise Concatenation)**.** *For strings $S \in \Sigma_S^n = S_1, S_2, \cdots, S_n$ and $I \in \Sigma_I^n = I_1, I_2, \cdots, I_n$, we define S* indexed by *I or* symbol-wise concatenation *of S and I as $S \times I = (S_1, I_1), (S_2, I_2), \cdots, (S_n, I_n)$. Note that $S \times I \in (\Sigma_S \times \Sigma_I)^n$.*

Definition 2.1.2 (Code Indexing)**.** *For string $I \in \Sigma_I^n$ and code $C \subseteq \Sigma_C^n$, we define C* indexed by *I, or $C \times I \subseteq (\Sigma_S \times \Sigma_I)^n$, as a code that is obtained by indexing each codeword of C with I.*

Edit Distance. Throughout this work, we frequently rely on the well-known *edit distance* metric defined as follows.

Definition 2.1.3 (Edit distance)**.** *The* edit distance *between two strings $S_1, S_2 \in \Sigma^*$, or* $\mathrm{ED}(S_1, S_2)$*, is the minimum number of insertions and deletions required to transform S_1 into S_2.*

It is easy to see that edit distance is a metric on any set of strings and in particular is symmetric and satisfies the triangle inequality property. Furthermore, $\mathrm{ED}(S_1, S_2) = |S_1|+|S_2|-2\cdot\mathrm{LCS}(S_1, S_2)$, where $\mathrm{LCS}(S_1, S_2)$ is the size of the longest common subsequence of S_1 and S_2.

2.2 Error Correcting Codes and InsDel Codes

Next, we give a quick summary of the standard definitions and formalism around error correcting codes.

Codes, Distance, Rate, and Half-Errors. An *error correcting code* C is an injective function which takes an input string $s \in (\Sigma')^{n'}$ over alphabet Σ' of length n' and generates a *codeword* $C(s) \in \Sigma^n$ of length n over alphabet Σ. The length n of a codeword is also called the *block length*. The two most important parameters of a code are its distance Δ and its rate R. The *rate* $R = \frac{n \log |\Sigma|}{n' \log |\Sigma'|}$ measures what fraction of bits in the codewords produced by C carries non-redundant information about the input. The *code distance* $\Delta(C) = \min_{s,s'} \Delta(C(s), C(s'))$ is simply the minimum Hamming distance between any two codewords. The *relative distance* $\delta(C) = \frac{\Delta(C)}{n}$ measures what fraction of output symbols need to be substitutions to transform one codeword into another.

It is easy to see that if a sender sends out a codeword $C(s)$ of code C with relative distance δ, a receiver can uniquely recover s if she receives a codeword in which less than a δ fraction of symbols are affected by an *erasure*, i.e., replaced by a special "?" symbol. Similarly, the receiver can uniquely recover the input s if less than $\delta/2$ *symbol substitutions*—in which a symbol is replaced by any other symbol from Σ—occurred. More generally, it is easy to see that the receiver can recover from any combination of k_e erasures and k_s substitutions as long as $k_e+2k_s < \delta n$. This motivates defining *half-errors* to measure both erasures and symbol substitutions where an erasure is counted as a single half-error and a symbol substitution is counted as two half-errors. In summary, any code of relative distance δ can tolerate any error pattern of less than δn half-errors.

Synchronization Errors. In addition to half-errors, we study *synchronization errors* which consist of *deletions*, that is, a symbol being removed without replacement, and *insertions*, where a new symbol from Σ is added anywhere. It is clear that synchronization errors are strictly harsher and more general than half-errors (see Section 3.1.1). The above formalism of codes, rate, and distance works equally well for synchronization errors if one replaces the Hamming distance with edit distance. Instead of measuring the number of symbol substitutions required to transform one string into another, edit distance measures the minimum number of insertions and deletions to do so. Similar to error correcting codes for Hamming-type errors, an error correcting code for insertions and deletions (or InsDel code for short) is defined as a subset $C \subseteq \Sigma^n$ for alphabet set Σ and block length n. The minimum distance of C is defined as the minimum edit distance between the codewords of

C, i.e.

$$\delta_C = \frac{\min_{x,y \in C} \mathrm{ED}(x,y)}{2n}.$$

We remark that the edit distance of two strings of length n can be as large as $2n$ and hence the normalizing divisor is $2n$. Further, the rate of the code C is defined as $r_C = \frac{\log |C|}{n \log |\Sigma|}$. An encoding function $\mathrm{Enc}_C : \Sigma^{nr} \to \Sigma^n$ for C is a bijective function that maps any string in Σ^{nr} to a member of C and a decoding function Dec_C is one that takes any string in $w \in \Sigma^n$ and returns the (unique) codeword that is within δn edit distance of w or $\perp$ if such codeword does not exist.

In this work we often consider *families* of codes, that are formally defined as follows.

Definition 2.2.1 (Family of Codes)**.** *A family of codes C with distance δ and rate r is defined as an infinite series of codes with increasing block length like $C_1, C_2, \cdots$ with distance δ and respective rates $r_1, r_2, \cdots$ where $\lim_{n\to\infty} r_i = r$.*

Efficient Codes. In addition to codes with a good minimum distance, one furthermore wants efficient algorithms for the encoding and error-correction tasks associated with the code. Throughout this thesis, we say a code is efficient if it has encoding and decoding algorithms running in time polynomial in terms of the block length. While it is often not hard to show that random codes exhibit a good rate and distance, designing codes which can be decoded efficiently is much harder. We remark that most codes which can efficiently correct for symbol substitutions are also efficient for half-errors. For insdel codes the situation is slightly different. While it remains true that any code that can uniquely be decoded from any $\delta(C)$ fraction of deletions can also be decoded from the same fraction of insertions and deletions [Lev65] doing so efficiently is often much easier for the deletion-only setting than the fully general insdel setting.

Chapter 3

Coding via Indexing with Synchronization Strings

In this chapter, we introduce *synchronization strings*, which provide a novel way to efficiently deal with synchronization errors, i.e., insertions and deletions, which are strictly more general and much harder to cope with than more commonly considered *Hamming-type errors*, i.e., symbol substitutions and erasures. For every $\varepsilon > 0$, synchronization strings allow to index a sequence with an $\varepsilon^{-O(1)}$ size alphabet such that one can **efficiently transform k synchronization errors into $(1+\varepsilon)k$ Hamming-type errors**. This powerful new technique has many applications. In this chapter, we focus on designing uniquely-decodable error correcting block codes for insertion-deletion channels.

While ECCs for both Hamming-type errors and synchronization errors have been intensely studied, the latter has largely resisted progress. As Mitzenmacher puts it in his 2009 survey [Mit09]: "*Channels with synchronization errors ... are simply not adequately understood by current theory. Given the near-complete knowledge we have for channels with erasures and errors ... our lack of understanding about channels with synchronization errors is truly remarkable.*" Indeed, it took until 1999 for the first insdel codes with constant rate, constant distance, and constant alphabet size to be constructed and only since 2016 are there constructions of constant rate insdel codes for asymptotically large noise rates. Even in the asymptotically large or small noise regimes, the codes proposed prior to this work were polynomially far from the optimal rate-distance tradeoff. This makes the understanding of insdel codes up to this work equivalent to what was known for regular ECCs after Forney introduced concatenated codes in his doctoral thesis 50 years ago.

A straightforward application of our synchronization strings based indexing method gives a simple black-box construction which **transforms any ECC into an equally efficient insdel code** with only a small increase in the alphabet size. This instantly transfers much of the highly developed understanding for regular ECCs into the realm of insdel codes. Most notably, for the complete noise spectrum we obtain efficient "near-MDS" insdel codes which get arbitrarily close to the optimal rate-distance tradeoff given by the Singleton bound. In particular, for any $\delta \in (0,1)$ and $\varepsilon > 0$, we give a family of insdel codes achieving a rate of $1 - \delta - \varepsilon$ over a constant size alphabet that efficiently corrects a δ fraction of insertions or deletions.

3.1 Introduction

This chapter introduces synchronization strings, a new combinatorial structure which allows efficient synchronization and indexing of streams under insertions and deletions. Synchronization strings and our indexing abstraction provide a powerful and novel way to deal with synchronization issues. They make progress on the issues raised above and have applications in a large variety of settings and problems. We have found applications in channel simulations, synchronization sequences [MBT10], interactive coding schemes [Gel17, KR13, Hae14, GHS14, GH14, GH17a], edit distance tree codes [BGMO17], and error correcting codes for insertion and deletions and suspect there will be many more. This chapter focuses on the last application, namely, designing efficient error correcting block codes over large alphabets for worst-case insertion-deletion channels.

The knowledge on efficient error correcting block codes for insertions and deletions, also called *insdel codes*, severely lags behind what is known for codes for Hamming errors. While Levenshtein [Lev65] introduced and pushed the study of such codes already in the 1960s it took until 1999 for Schulman and Zuckerman [SZ99] to construct the first insdel codes with constant rate, constant distance, and constant alphabet size. Works of Guruswami et al. [GW17, GL16] in 2015 and 2016 gave the first constant rate insdel codes for asymptotically large noise rates via list decoding. These codes are, however, polynomially far from optimal in their rate or decodable distance respectively. In particular, they achieve a rate of $\Omega(\epsilon^5)$ for a relative distance of $1-\epsilon$ or a relative distance of $O(\epsilon^2)$ for a rate of $1-\epsilon$, for asymptotically small $\epsilon > 0$ (see Section 3.1.5 for a more detailed discussion of related work).

This chapter essentially closes this line of work by designing efficient "near-MDS" insdel codes which approach the optimal rate-distance trade-off given by the Singleton bound. We prove that for any $0 \leq \delta < 1$ and any constant $\varepsilon > 0$, there is an efficient family of insdel codes over a constant size alphabet with rate $1-\delta-\varepsilon$ which can be uniquely and efficiently decoded from any δ fraction of insertions and deletions. The code construction takes polynomial time; and encoding and decoding can be done in linear and quadratic time, respectively. More formally, we achieve the following theorem using the notion of the edit distance of two given strings as the minimum number of insertions and deletions required to convert one of them to the other one.

Theorem 3.1.1. *For any $\varepsilon > 0$ and $\delta \in (0,1)$ there exists an encoding map $\mathrm{Enc} : \Sigma^k \to \Sigma^n$ and a decoding map $\mathrm{Dec} : \Sigma^* \to \Sigma^k$, such that, if $\mathrm{EditDistance}(\mathrm{Enc}(m), x) \leq \delta n$ then $\mathrm{Dec}(x) = m$. Further, $\frac{k}{n} > 1-\delta-\varepsilon$, $|\Sigma| = f(\varepsilon)$, and Enc and Dec are explicit and can be computed in linear and quadratic time in n.*

This code is obtained via a black-box construction which **transforms any ECC into an equally efficient insdel code** with only a small increase in the alphabet size. This transformation, which is a straightforward application of our new synchronization strings based indexing method, is so simple that it can be summarized in one sentence:

For *any* efficient ECC with alphabet bit size $\frac{\log \varepsilon^{-1}}{\varepsilon}$, attaching to every codeword, symbol by symbol, a random or suitable pseudorandom string over

an alphabet of bit size $\log \varepsilon^{-1}$ results in an efficient insdel code with a rate and decodable distance that are changed by at most ε.

Far beyond just implying Theorem 3.1.1, this enables us to instantly transfer much of the highly developed understanding for regular ECCs into the realm of insdel codes.

Theorem 3.1.1 is obtained by using the "near-MDS" expander codes of Guruswami and Indyk [GI05] as a base ECC. These codes generalize the linear time codes of Spielman [Spi96] and can be encoded and decoded in linear time. Our simple encoding strategy, as outlined above, introduces essentially no additional computational complexity during encoding. Our quadratic time decoding algorithm, however, is slower than the linear time decoding of the base codes from [GI05] but still pretty fast. In particular, a quadratic time decoding for an insdel code is generally very good given that, in contrast to Hamming codes, even computing the distance between the received and the sent/decoded string is an edit distance computation. Edit distance computations, in general, do not run in subquadratic time, which is not surprising given the SETH-conditional lower bounds [BI18]. We, however, will later enhance the decoding procedure to rub in near-linear time in Chapter 9. Also, for the settings of insertion-only and deletion-only errors, we achieve analogs of Theorem 3.1.1 with linear decoding time complexities in this chapter.

3.1.1 High-level Overview, Intuition and Overall Organization

While extremely powerful, the concept and idea behind synchronization strings is easily demonstrated. In this section, we explain the high-level approach taken and provide intuition for the formal definitions and proofs to follow. This section also explains the overall organization of the rest of the chapter.

Synchronization Errors and Half-Errors

Consider a stream of symbols over a large but constant size alphabet Σ in which some constant fraction δ of symbols is corrupted. There are two basic types of corruptions we will consider, Hamming-type errors and synchronization errors. Hamming-type errors consist of *erasures*, that is, a symbol being replaced with a special "?" symbol indicating the erasure, and *symbol substitutions* in which a symbol is replaced with any other symbol in Σ. In this thesis, we measure Hamming-type errors in terms of *half-errors*. The wording half-error comes from the realization that, when it comes to code distances, erasures are half as bad as symbol substitution. An erasure is thus counted as one half-error while a symbol substitution counts as two half-errors. *Synchronization errors* consist of *deletions*, that is, a symbol being removed without replacement, and *insertions*, where a new symbol from Σ is added anywhere.

It is clear that **synchronization errors are strictly more general and harsher than half-errors**. In particular, any symbol substitution, worth two half-errors, can also be achieved via a deletion followed by an insertion. Any erasure can furthermore be interpreted as a deletion together with the often very helpful extra information where this deletion took place. This makes synchronization errors at least as hard as half-errors. The

real problem that synchronization errors bring with them, however, is that they cause sending and receiving parties to become "out of sync". This easily changes how received symbols are interpreted and makes designing codes or other systems tolerant to synchronization errors an inherently difficult and significantly less well understood problem.

Indexing and Synchronization Strings: Reducing Synchronization Errors to Half-Errors

There is a simple folklore strategy, which we call *indexing*, that avoids these synchronization problems: Simply enhance any element with a time stamp or element count. More precisely, consecutively number the elements and attach this position count or *index* to each element of the stream. Now, if we only deal with deletions, it is clear that the position of any deletion is easily identified via a missing index, thus transforming it into an erasure. Insertions can be handled similarly by treating any stream index which is received more than once as erased. If both insertions and deletions are allowed, one might still have elements with a spoofed or substituted value caused by a deletion of the indexed symbol which is then replaced by a different symbol with the same index inserted. This, however, requires two insdel errors. Generally, this *trivial indexing strategy* can be seen to successfully *transform any k synchronization errors into at most k half-errors.*

In many applications, however, this trivial indexing cannot be used because having to attach a $\log n$ bit long index description to each element of an n-long stream is prohibitively costly. Consider for example an error correcting code of constant rate R over some potentially large but nonetheless constant size alphabet Σ, which encodes $nR\log|\Sigma|$ bits into n symbols from Σ. Increasing Σ by a factor of n to allow each symbol to carry its $\log n$ bit index would destroy the desirable property of having an alphabet which is independent from the block length n and would furthermore reduce the rate of the code from R to $\Theta(\frac{R}{\log n})$, which approaches zero for large block lengths. For streams of unknown or infinite length such problems become even more pronounced.

This is where *synchronization strings* come to the rescue. Essentially, synchronization strings allow one to **index every element in an infinite stream using only a constant size alphabet** while achieving an arbitrarily good approximate reduction from synchronization errors to half-errors. In particular, using synchronization strings ***k* synchronization errors can be transformed into at most $(1+\varepsilon)k$ half-errors using an alphabet of size independent of the stream length** and in fact only polynomial in $\frac{1}{\varepsilon}$. Moreover, these synchronization strings have simple constructions and fast and easy repositioning procedures–i.e., algorithms that guess the original position of symbols using the indexed synchronization string.

Attaching our synchronization strings to the codewords of any efficient error correcting code which efficiently tolerates the usual symbol substitutions and erasures, transforms any such code into an efficiently decodable insdel code while only requiring a negligible increase in the alphabet size. This allows the decades of intense research in coding theory for Hamming-type errors to be transferred into the much harder and less well-understood insertion-deletion setting.

3.1.2 Synchronization Strings: Definition, Construction, and Decoding

Next, we briefly motivate and explain how one arrives at the natural definition of these index sequences over a finite alphabet and what intuition lies behind their efficient constructions and decoding procedures.

Suppose that a sender has attached the symbols if an index sequence S to elements of a communication stream and consider the time at which the receiver has received a corrupted sequence of the first t index descriptors, i.e., a corrupted version of t-long prefix of S. As the receiver tries to guess or *decode* the true position of the last received symbol at this time, it should naturally consider all index symbols received so far and find the "most plausible" prefix of S. This suggests that the prefix of length l of a synchronization string S acts as a codeword for the position l and one should think of the set of prefixes of S as a code associated with the synchronization string S. Naturally, one would want such a code to have good distance properties between any two codewords under some distance measure. While edit distance, i.e., the number of insertions and deletions needed to transform one string into another seems like the right notion of distance for insdel errors in general, the prefix nature of the codes under consideration will guarantee that codewords for indices l and $l' > l$ will have edit distance exactly $l' - l$. This implies that even two very long codewords only have a tiny edit distance. On the one hand, this precludes synchronization codes with a large relative edit distance between its codewords. On the other hand, one should see this phenomenon as simply capturing the fact that at any time a simple insertion of an incorrect symbol carrying the correct next index symbol will lead to an unavoidable decoding error. Given this natural and unavoidable sensitivity of synchronization codes to recent errors, it makes sense to, instead, use a distance measure which captures the recent density of errors. In this spirit, we suggest the definition of a new (to our knowledge) string distance measure which we call *relative suffix distance*, which intuitively measures the worst fraction of insdel errors to transform suffixes, i.e., recently sent parts of two strings, into each other. This natural measure, in contrast to a similar measure defined in [BGMO17], turns out to induce a metric space on any set of strings.

With these natural definitions for an induced set of codewords and a natural distance metric associated with any such set, the next task is to design a string S for which the set of codewords has as large of a minimum pairwise distance as possible. When looking for (infinite) sequences that induce such a set of codewords, and thus can be successfully used as index strings, it becomes apparent that one is looking for highly irregular and non-self-similar strings over a fixed alphabet Σ. It turns out that the correct definition to capture these desired properties, which we call the ε-synchronization property, states that any two neighboring intervals of S with total length l should require at least $(1 - \varepsilon)l$ insertions and deletions to transform to one another, where $\varepsilon \geq 0$. A simple calculation shows that this clean property also implies a large minimum relative suffix distance between any two codewords. Not surprisingly, random strings essentially satisfy this ε-synchronization property, except for local imperfections of self-similarity, such as, symbols repeated twice in a row, which would naturally occur in random sequences about every $|\Sigma|$ positions. This allows us to use the probabilistic method and the general Lovász local lemma to prove the

existence ε-synchronization strings. This also leads to an efficient randomized construction.

Finally, decoding any string to the closest codeword, i.e., the prefix of the synchronization string S with the smallest relative suffix distance, can be easily done in polynomial time because the size of the set of codewords associated with the synchronization string S is linear and not exponential in n and suffix distance computations (to each codeword individually) can be done in polynomial time as they essentially consist of edit distance computations between suffixes of the two input strings.

3.1.3 More Sophisticated Decoding Procedures

All this provides an indexing solution which transforms any k synchronization errors into at most $(5+\varepsilon)k$ half-errors. This already leads to insdel codes which achieve a rate approaching $1-5\delta$ for any δ fraction of insdel errors with $\delta < \frac{1}{5}$. While this is already a drastic improvement over the previously best $1-O(\sqrt{\delta})$ rate codes from [GL16], which worked only for sufficiently small δ, it is a far less strong result than the near-MDS codes we promised in Theorem 3.1.1 for every $\delta \in (0,1)$.

We were able to slightly improve upon the above strategy by considering an alternative to the relative suffix distance measure, which we call relative suffix pseudo distance (RSPD). RSPD was introduced in [BGMO17] and, while neither being symmetric nor satisfying the triangle inequality, can act as a pseudo-distance in the minimum-distance decoder. For any set of $k = k_i + k_d$ insdel errors consisting of k_i insertions and k_d deletions, this improved indexing solution leads to no more than $(1+\varepsilon)(3k_i + k_d)$ half-errors. This already implies near-MDS codes for deletion-only setting but still falls short for general insdel errors. We leave open the question whether an improved pseudo-distance definition can achieve an indexing solution with $(1+\varepsilon)k$ half-errors.

In order to achieve our main theorem, we developed an different strategy. Fortunately, it turned out that achieving a better indexing solution and the desired insdel codes does not require any changes to the definition of synchronization strings, the indexing approach itself, or the encoding scheme but solely required a very different decoding strategy. In particular, instead of guessing the position of symbols in a streaming manner we consider more global, offline repositioning algorithms. We provide several such repositioning algorithms in Section 3.5. In particular, we give a simple global repositioning algorithm, for which the number of misdecodings goes to zero as the parameter ε of the utilized ε-synchronization string goes to zero, irrespectively of how many insdel errors are applied.

Our global repositioning algorithms crucially build on another key-property which we prove holds for any ε-synchronization string S, namely that there is no monotone matching between S and itself which mismatches more than an ε-fraction of indices. Besides being used in our proofs, considering this ε-self-matching property has another advantage. We show that this property is achieved easier than the full ε-synchronization property and that indeed a random string satisfies it with good probability. This means that, in the context of error correcting codes, one can even use a simple uniformly random string as a "synchronization string". Lastly, we show that even an $n^{-O(1)}$-approximate $O\left(\frac{\log n}{\log(1/\varepsilon)}\right)$-wise independent random string satisfies the desired ε-self-matching property which, using

the celebrated small sample space constructions from [NN93] also leads to a deterministic polynomial time construction for the index string.

Lastly, we provide simpler and faster global repositioning algorithms for the setting of deletion-only and insertion-only errors. These algorithms are essentially greedy algorithms which run in linear time. They furthermore guarantee that their position guessing is error-free, i.e., they only output "I don't know" for some indices but never produce an incorrectly decoded index. Such decoding schemes have the advantage that one can use them in conjunction with error correcting codes that efficiently recover from erasures (and not necessarily from symbol substitutions).

3.1.4 Organization of this Chapter

The organization of this chapter closely follows the flow of the high-level description above. We start by giving more details on related work in Section 3.1.5. In Section 3.2, we formalize the indexing problem and solutions to it. Section 7.5 shows how any solution to the indexing problem can be used to transform any regular error correcting codes into an insdel code. Section 3.4 introduces the relative suffix distance and ε-synchronization strings, proves the existence of ε-synchronization strings and provides an efficient construction. Section 3.4.2 shows that the minimum suffix distance decoder is efficient and leads to a good indexing solution. We elaborate on the connection between ε-synchronization strings and the ε-self-matching property in Section 3.5.1, introduce an efficient deterministic construction of ε-self matching strings in Section 3.5.2, and provide our improved repositioning algorithms in the remainder of Section 3.5.

3.1.5 Related Work

Shannon was the first to systematically study reliable communication. He introduced random error channels, defined information quantities, and gave probabilistic existence proofs of good codes. Hamming was the first to look at worst-case errors and code distances as introduced above. Simple counting arguments on the volume of balls around codewords given in the 50's by Hamming and Gilbert-Varshamov produce simple bounds on the rate of q-ary codes with relative distance δ. In particular, they show the existence of codes with relative distance δ and rate at least $1 - H_q(\delta)$ where $H_q(x) = x \log_q(q-1) - x \log_q x - (1-x) \log_q(1-x)$ is the q-ary entropy function. This means that for any $\delta < 1$ and $q = \omega(1/\delta)$ there exists codes with distance δ and rate approaching $1 - \delta$. Concatenated codes and the generalized minimum distance decoding procedure introduced by Forney in 1966 led to the first codes which could recover from constant error fractions $\delta \in (0, 1)$ while having polynomial time encoding and decoding procedures. The rate achieved by concatenated codes for large alphabets with sufficiently small distance δ comes out to be $1 - O(\sqrt{\delta})$. On the other hand, for δ sufficiently close to one, one can achieve a constant rate of $O(\delta^2)$. Algebraic geometry codes suggested by Goppa in 1975 later led to error correcting codes which, for every $\varepsilon > 0$, achieve the optimal rate of $1-\delta-\varepsilon$ with an alphabet size polynomial in ε while being able to efficiently correct from any δ fraction of half-errors [TV91].

While this answered the most basic questions, research since then has developed a tremendously powerful toolbox and selection of explicit codes. It attests to the importance of error correcting codes that over the last several decades this research direction has developed into the incredibly active field of coding theory with hundreds of researchers studying and developing better codes. A small and highly incomplete subset of important innovations include rateless codes, such as, LT codes [Lub02], which do not require to fix a desired distance at the time of encoding, explicit expander codes [Spi96, GI05] which allow linear time encoding and decoding, polar codes [GX15, GV15] which can approach Shannon's capacity polynomially fast, network codes [LYC03] which allow intermediate nodes in a network to recombine codewords, and efficiently list decodable codes [GR08] which allow to list-decode codes of relative distance δ up to a fraction of about δ symbol substitutions.

While error correcting codes for insertions and deletions have also been intensely studied, our understanding of them is much less well developed. We refer to the 2002 survey by Sloan [Slo02] on single-deletion codes, the 2009 survey by Mitzenmacher [Mit09] on codes for random deletions and the most general 2010 survey by Mercier et al. [MBT10] for the extensive work done around codes for synchronization errors and only mention the results most closely related to Theorem 3.1.1 here: Insdel codes were first considered by Levenshtein [Lev65] and since then many bounds and constructions for such codes have been given. However, while essentially the same volume and sphere packing arguments as for regular codes show that there exist families of insdel codes capable of correcting a fraction δ of insdel errors with rate approaching $1 - \delta$, no efficient constructions anywhere close to this rate-distance tradeoff are known. Even the construction of efficient insdel codes over a constant alphabet with any (tiny) constant relative distance and any (tiny) constant rate had to wait until Schulman and Zuckerman gave the first such code in 1999 [SZ99]. Over the couple of years preceding this work, Guruswami et al. provided new codes improving over this state-of-the-art in the asymptotically small or large noise regimes by giving the first codes which achieve a constant rate for noise rates going to one and codes which provide a rate going to one for an asymptotically small noise rate. In particular, [GW17] gave the first efficient codes codes over fixed alphabets to correct a deletion fraction approaching 1, as well as efficient binary codes to correct a small constant fraction of deletions with rate approaching 1. These codes could, however, only be efficiently decoded for deletions and not insertions. A follow-up work gave new and improved codes with similar rate-distance tradeoffs which can be efficiently decoded from insertions and deletions [GL16]. In particular, these codes achieve a rate of $\Omega((1-\delta)^5)$ and $1 - \tilde{O}(\sqrt{\delta})$ while being able to efficiently recover from a δ fraction of insertions and deletions in high-noise and high-rate regimes respectively. These works put the state-of-the-art for error correcting codes for insertions and deletions prior to this work pretty much equal to what was known for regular error correcting codes 50 years ago, after Forney's 1965 doctoral thesis.

3.2 The Indexing Problem

In this section, we formally define the indexing problem. In a nutshell, this problem is that of sending a suitably chosen string S of length n over an insertion-deletion channel such that the receiver will be able to figure out the original position of most of the symbols he receives correctly. This problem can be trivially solved by sending the string $S = 1, 2, \ldots, n$ over the alphabet $\Sigma = \{1, \ldots, n\}$ of size n. This way, the original position of every received symbol is equal to its value. We will provide an interesting solution to the indexing problem that does almost as well while using a finite size alphabet. While very intuitive and simple, the formalization of this problem and its solutions enables an easy use in many applications.

Before giving a formal presentation, we introduce the *string matching* notion from [BGMO17].

Definition 3.2.1 (String matching)**.** *Suppose that c and c' are two strings in Σ^* and $*$ is a symbol not included in Σ. Further, assume that there exist two strings τ_1 and τ_2 in $(\Sigma \cup \{*\})^*$ such that $|\tau_1| = |\tau_2|$, $\mathrm{del}(\tau_1) = c$, $\mathrm{del}(\tau_2) = c'$, and $\tau_1[i] \approx \tau_2[i]$ for all $i \in \{1, \ldots, |\tau_1|\}$. Here,* del *is a function that deletes every $*$ in the input string and $a \approx b$ if $a = b$ or one of a or b is $*$. Then, we say that $\tau = (\tau_1, \tau_2)$ is a* string matching *between c and c' (denoted by $\tau : c \to c'$). We furthermore denote with $\mathrm{sc}(\tau_i)$ the number of $*$'s in τ_i.*

Note that the *edit distance* between strings $c, c' \in \Sigma^*$ is exactly equal to

$$\min_{\tau : c \to c'} \{\mathrm{sc}(\tau_1) + \mathrm{sc}(\tau_2)\}.$$

We now formally define an (n, δ)-indexing problem where n denotes the number of symbols which are being sent and δ denotes the maximum fraction of symbols that can be inserted or deleted. We further call the string S of length n the *index string*. Lastly, we describe the effect of the $n\delta$ worst-case insertions and deletions which transform S into the related string S_τ in terms of a string matching τ. In particular, $\tau = (\tau_1, \tau_2)$ is the string matching from S to S_τ such that $\mathrm{del}(\tau_1) = S$, $\mathrm{del}(\tau_2) = S_\tau$, and for every k

$$(\tau_1[k], \tau_2[k]) = \begin{cases} (S[i], *) & \text{if } S[i] \text{ is deleted} \\ (S[i], S_\tau[j]) & \text{if } S[i] \text{ is delivered as } S_\tau[j] \\ (*, S_\tau[j]) & \text{if } S_\tau[j] \text{ is inserted} \end{cases}$$

where $i = |\mathrm{del}(\tau_1[1, k])|$ and $j = |\mathrm{del}(\tau_2[1, k])|$. (See Definition 3.2.1 for the definition of del)

Definition 3.2.2 ((n, δ)-Indexing Solution)**.** *The pair $(S, \mathcal{D}_S)$ consisting of the* index string *$S \in \Sigma^n$ and the* repositioning algorithm *$\mathcal{D}_S$ is called a solution for (n, δ)-indexing problem over alphabet Σ if, for any set of $n\delta$ insertions and deletions represented by τ which alters S to a string S_τ, the algorithm $\mathcal{D}_S(S_\tau)$ outputs, for every symbol in S_τ, either $\perp$ or an index between 1 and n, i.e., $\mathcal{D}_S : \Sigma^{|S_\tau|} \to ([1..n] \cup \perp)^{|S_\tau|}$.*

The $\perp$ symbol here represents an "I don't know" response by the repositioning algorithm while a numerical output j for the ith symbol of S_τ should be interpreted as the algorithm

guessing that $S_\tau[i]$ was at position j in string S prior to going through the insertion-deletion channel. We frequently refer to this procedure of guessing the position of the ith index symbol as *decoding index i*. One seeks algorithms that correctly decode as many indexes as possible. Naturally, one can only *correctly decode* index symbols that were *successfully transmitted*. We give formal definitions of both notions here.

Definition 3.2.3 (Correctly Decoded Index Symbol)**.** *An (n, δ)-indexing solution $(S, \mathcal{D}_S)$ decodes index j correctly under τ if $\mathcal{D}_S(S_\tau)$ outputs i for the jth received symbol and there exists a k such that $i = |\mathrm{del}(\tau_1[1, k])|, j = |\mathrm{del}(\tau_2[1, k])|, \tau_1[k] = S[i],$ and $\tau_2[k] = S_\tau[j]$.*

We remark that this definition counts any $\perp$ response as an incorrect decoding.

Definition 3.2.4 (Successfully Transmitted Symbol)**.** *For string S_τ, which was derived from an index string S via $\tau = (\tau_1, \tau_2)$, we call the jth symbol $S_\tau[j]$ successfully transmitted if it stems from a symbol coming from S, i.e., if there exists a k such that $|\mathrm{del}(\tau_2[1, k])| = j$ and $\tau_1[k] = \tau_2[k]$.*

We now propose a way to measure the quality of an (n, δ)-indexing solution by counting the maximum number of misdecoded symbols among those that were successfully transmitted. Note that the trivial indexing strategy with $S = 1, \ldots, n$ which outputs for each symbol the symbol itself has no misdecodings. One can therefore also interpret our definition of quality as capturing how far from this ideal solution a given indexing solution is (stemming likely from the smaller alphabet which is used for S).

Definition 3.2.5 (Misdecodings of an (n, δ)-Indexing Solution)**.** *We say that an (n, δ)-indexing solution has at most k misdecodings if for any τ corresponding to at most $n\delta$ insertions and deletions, the number of successfully transmitted index symbols that are incorrectly decoded is at most k.*

Now, we introduce two further useful properties that an (n, δ)-indexing solution might have.

Definition 3.2.6 (Error-free Solution)**.** *We call $(S, \mathcal{D}_S)$ an error-free (n, δ)-indexing solution if for any error pattern τ, and for any element of S_τ, the repositioning algorithm $\mathcal{D}_S$ either outputs $\perp$ or correctly decodes that element. In other words, the repositioning algorithm never makes an incorrect guess, even for symbols which are inserted by the channel–it may just output $\perp$ for some of the successfully transmitted symbols.*

It is noteworthy that error-free solutions are essentially only obtainable when dealing with insertion-only or deletion-only settings. In both cases, the trivial solution with $S = 1, \cdots, n$ is error-free. We will introduce indexing solutions which provide this nice property, even over a smaller alphabet, and show how being error-free can be useful in the context of error correcting codes.

Lastly, another very useful property of some (n, δ)-indexing solutions is that their decoding process operates in a streaming manner, i.e, the repositioning algorithm guesses the position of $S_\tau[j]$ independently of $S_\tau[j']$ where $j' > j$. While this property is not particularly useful for the error correcting block code application put forward in this chapter, it

is an extremely important and strong property which is crucial in several applications we know of, such as, rateless error correcting codes, channel simulations, interactive coding, edit distance tree codes, and other settings.

Definition 3.2.7 (Streaming Solutions)**.** *We call $(S, \mathcal{D}_S)$ a streaming solution if the output of $\mathcal{D}_S$ for the ith element of the received string S_τ only depends on $S_\tau[1, i]$.*

Again, the trivial solution for (n, δ)-indexing problem over an alphabet of size n with zero misdecodings can be made streaming by outputting for every received symbols the received symbol itself as the guessed position. This solution is also error-free for the deletion-only setting but not error-free for the insertion-only setting. In fact, it is easy to show that an algorithm cannot be both streaming and error-free in any setting which allows insertions.

Overall, the important characteristics of an (n, δ)-indexing solution are (a) its alphabet size $|\Sigma|$, (b) the number of misdecodings it might bring about, (c) time complexity of the repositioning algorithm $\mathcal{D}_S$, (d) time complexity of constructing the index string S (preprocessing), (e) whether the algorithm works for the insertion-only, the deletion-only or the full insdel setting, and (f) whether the algorithm satisfies the streaming or error-free properties. Table 3.1 gives a summary over the different solutions for the (n, δ)-indexing problem we give in this chapter. The repositioning algorithm in all these solutions are deterministic and the index string is over an alphabet of size $\varepsilon^{-O(1)}$ for parameter $\varepsilon > 0$ that can be chosen arbitrarily small.

Algorithm	Type	Misdecodings	Error-free	Streaming	$\mathcal{D}_S(\cdot)$ Complexity
Section 3.4.2	ins/del	$(2+\varepsilon) \cdot n\delta$		✓	$O(n^4)$
Section 3.5.3	ins/del	$3\sqrt{\varepsilon} \cdot n$			$O\left(n^2/\sqrt{\varepsilon}\right)$
Section 3.5.4	del	$\varepsilon \cdot n\delta$		✓	$O(n)$
Section 3.5.5	ins	$(1+\varepsilon) \cdot n\delta$	✓		$O(n)$
Section 3.5.5	del	$\varepsilon \cdot n\delta$	✓		$O(n)$
Section 3.5.7	ins/del	$(1+\varepsilon) \cdot n\delta$		✓	$O(n^4)$

Table 3.1: Properties and quality of (n, δ)-indexing solutions with S being an ε-synchronization string. The alphabet size of string S is $\varepsilon^{-O(1)}$.

3.3 Insdel Codes via Indexing Solutions

Next, we show how a good (n, δ)-indexing solution $(S, \mathcal{D}_S)$ over alphabet Σ_S allows one to transform any regular ECC $\mathcal{C}$ with block length n over alphabet $\Sigma_{\mathcal{C}}$ which can efficiently correct half-errors, i.e., symbol substitutions and erasures, into a good insdel code over alphabet $\Sigma = \Sigma_{\mathcal{C}} \times \Sigma_S$.

To this end, we simply attach S symbol-by-symbol to every codeword of $\mathcal{C}$, i.e., obtain the code $\mathcal{C}_{out} = \{x \times S | x \in \mathcal{C}\}$. On the decoding end, we first guess the original positions of the symbols arrived using only the index portion of each received symbol and rearrange

them accordingly. Positions where zero or multiple symbols are mapped to are considered as '?', i.e., ambiguous. We will refer to this procedure as *the rearrangement procedure*. Finally, the decoding algorithm $\mathcal{D}_{\mathcal{C}}$ for $\mathcal{C}$ is used over this rearranged string to finish decoding. These two straightforward algorithms are formally described as Algorithm 1 and Algorithm 2.

Theorem 3.3.1. *If $(S, \mathcal{D}_S)$ guarantees k misdecodings for the (n, δ)-indexing problem, then the rearrangement procedure recovers the sent codeword up to $n\delta + 2k$ half-errors, i.e., half-error distance of the codeword sent and the one recovered by the rearrangement procedure is at most $n\delta + 2k$. If $(S, \mathcal{D}_S)$ is error-free, the rearrangement procedure recovers the sent codeword up to $n\delta + k$ half-errors.*

Proof. Consider a set of insertions and deletions described by τ consisting of D_τ deletions and I_τ insertions. Note that among n index symbols, at most D_τ were deleted and less than k are decoded incorrectly. Therefore, at least $n - D_\tau - k$ index symbols are decoded correctly. Thus, if the rearrangement procedure only included correctly decoded indices for successfully transmitted symbols, the output would have contained up to $D_\tau + k$ erasures and no symbol substitutions, resulting into a total of $D_\tau + k$ half-errors. However, any symbol which is being incorrectly decoded or inserted may cause a correctly decoded index to become an erasure by making it appear multiple times or change one of the initial $D_\tau + k$ erasures into a substitution error by making the rearrangement procedure mistakenly identify an index at that position. Overall, this can increase the number of half-errors by at most $I_\tau + k$ to a total of at most $D_\tau + k + I_\tau + k = D_\tau + I_\tau + 2k \leq n\delta + 2k$ half-errors.

For error-free indexing solutions, misdecodings only consist of $\perp$ symbols and therefore only result in erasures (one half-error). Thus, the number of incorrect indices is I_τ instead of $I_\tau + k$ leading to the reduced number of half-errors in this case. □

This makes it clear that applying an ECC $\mathcal{C}$ which is resilient to $n\delta + 2k$ half-errors on top of an (n, δ)-indexing solution enables the receiver side to fully recover m.

Algorithm 1 Insertion-Deletion Encoder using $\mathcal{C}$ and $(S, \mathcal{D}_S)$

Input: m

1: $\tilde{m} = \mathcal{E}_{\mathcal{C}}(m)$

Output: $\tilde{m} \times S$

Next, we formally state how a good (n, δ)-indexing solution $(S, \mathcal{D}_S)$ over alphabet Σ_S allows one to transform any regular ECC $\mathcal{C}$ with block length n over alphabet $\Sigma_{\mathcal{C}}$ which can efficiently correct half-errors, i.e., symbol substitutions and erasures, into a good insdel code over alphabet $\Sigma = \Sigma_{\mathcal{C}} \times \Sigma_S$. The following Theorem is a corollary of Theorem 3.3.1 and the definition of the rearrangement procedure.

Theorem 3.3.2. *Given an (efficient) (n, δ)-indexing solution $(S, \mathcal{D}_S)$ over alphabet Σ_S with at most k misdecodings, and repositioning time complexity $T_{\mathcal{D}_S}$ and an (efficient) ECC $\mathcal{C}$ over alphabet $\Sigma_{\mathcal{C}}$ with rate $R_{\mathcal{C}}$, encoding complexity $T_{\mathcal{E}_{\mathcal{C}}}$, and decoding complexity $T_{\mathcal{D}_{\mathcal{C}}}$ that corrects up to $n\delta + 2k$ half-errors, one can obtain an insdel code by indexing*

Algorithm 2 Insertion-Deletion Decoder using $\mathcal{C}$ and $(S, \mathcal{D}_S)$

Input: $y = \tilde{m}' \times S'$

1: $Dec \leftarrow \mathcal{D}_S(S')$
2: **for** i $= 1$ to n **do**
3: **if** there is a unique j for which $Dec[j] = i$ **then**
4: $\tilde{m}[i] = \tilde{m}'[j]$
5: **else**
6: $\tilde{m}[i] = ?$

Output: $\mathcal{D}_{\mathcal{C}}(\tilde{m})$

codewords of $\mathcal{C}$ with S that can (efficiently) decode from up to $n\delta$ insertions and deletions. The rate of this code is at least

$$\frac{R_{\mathcal{C}}}{1 + \frac{\log|\Sigma_S|}{\log|\Sigma_{\mathcal{C}}|}}$$

The encoding complexity remains $T_{\mathcal{E}_{\mathcal{C}}}$, the decoding complexity is $T_{\mathcal{D}_{\mathcal{C}}} + T_{\mathcal{D}_S}$ and the preprocessing complexity of constructing the code is the complexity of constructing $\mathcal{C}$ and S. Furthermore, if $(S, \mathcal{D}_S)$ is error-free, then choosing a $\mathcal{C}$ which can recover only from $n\delta + k$ erasures is sufficient to produce a code with same qualities.

Proof. Theorem 3.3.1 directly implies the distance quality. The encoding time complexity follows from the fact that concatenating the codeword with the index string takes linear time. The decoding time complexity is simply the sum of the running time of the algorithm $\mathcal{D}_S$ and the decoding complexity of code C. Finally, given that this transformation keeps the number of codewords as that of C and only increases the alphabet size, the rate of the resulting code is

$$R = \frac{|C|}{n \log|\Sigma_C \times \Sigma_S|} = \frac{|C|}{n(\log|\Sigma_C| + \log|\Sigma_S|)} = \frac{R_{\mathcal{C}}}{1 + \frac{\log|\Sigma_S|}{\log|\Sigma_{\mathcal{C}}|}}.$$

□

Note that if one chooses $\Sigma_{\mathcal{C}}$ such that $\frac{\log|\Sigma_S|}{\log|\Sigma_{\mathcal{C}}|} << 1$, the rate loss due to the attached symbols will be negligible. Using the observations outlined so far, one can obtain Theorem 3.1.1 as a consequence of Theorem 3.3.2.

3.3.1 Proof of Theorem 3.1.1

To prove this, we make use of the indexing solution that we will present later in Section 3.5.3. As outlined in Table 3.1, for any $\delta, \varepsilon' \in (0, 1)$, Section 3.5.3 provides an indexing solution with $3n\sqrt{\varepsilon'}$ misdecodings and $O(n^2/\sqrt{\varepsilon'})$ repositioning time complexity. The alphabet size of the string in this solution is $\varepsilon'^{-O(1)}$. We also make use of the following near-MDS expander codes from [GI05].

Theorem 3.3.3 (Guruswami and Indyk [GI05, Theorem 3]). *For every r, $0 < r < 1$, and all sufficiently small $\varepsilon > 0$, there exists an explicitly specified family of GF(2)-linear (also called additive) codes of rate r and relative distance at least $(1 - r - \varepsilon)$ over an alphabet of size $2^{O(\varepsilon^{-4} r^{-1} \log(1/\varepsilon))}$ such that codes from the family can be encoded in linear time and can also be (uniquely) decoded in linear time from a fraction e of errors and s of erasures provided $2e + s \leq (1 - r - \varepsilon)$.*

Given the δ and ε from the statement of Theorem 3.1.1 we choose $\varepsilon' = \frac{\varepsilon^2}{18^2}$ and consider the (n, δ)-indexing solution $(S, \mathcal{D}_S)$ as given in Section 3.5.3 (see line 2 of Table 3.1) which guarantees no more than $3n\sqrt{\varepsilon'} = \frac{n\varepsilon}{6}$ misdecodings. We then choose a near-MDS expander code $\mathcal{C}$ from [GI05] (Theorem 3.3.3) which can efficiently correct up to $\delta_{\mathcal{C}} = \delta + \frac{\varepsilon}{3}$ half-errors and has a rate of $R_{\mathcal{C}} > 1 - \delta_{\mathcal{C}} - \frac{\varepsilon}{3} = 1 - \delta - \frac{2\varepsilon}{3}$ over an alphabet $\Sigma_{\mathcal{C}}$ of size $\exp(\varepsilon^{-O(1)})$ that satisfies $\log |\Sigma_{\mathcal{C}}| \geq \frac{3 \log |\Sigma_S|}{\varepsilon}$. This ensures that the final rate is indeed at least $\frac{R_{\mathcal{C}}}{1 + \frac{\log |\Sigma_S|}{\log |\Sigma_{\mathcal{C}}|}} \geq R_{\mathcal{C}} - \frac{\log |\Sigma_S|}{\log |\Sigma_{\mathcal{C}}|} = 1 - \delta - 3\frac{\varepsilon}{3}$ and the fraction of insdel errors that can be efficiently corrected is $\delta_{\mathcal{C}} - 2\frac{\varepsilon}{6} = \delta$. The encoding and decoding complexities follow Theorems 3.3.2 and 3.3.3 and the time complexity of the indexing solution as indicated in line 2 of Table 3.1. □

Theorem 3.1.1 is clearly optimal in its tradeoff between rate and efficiently decodable distance. As discussed in Section 3.1, its linear and quadratic encoding and decoding times are also optimal or hard to improve upon. (See Chapter 9 for an improved near-linear time decoding algorithm.) The only parameter that could be tightened is the dependence of the alphabet bit size on the parameter ε, which characterizes how close a code is to achieving an optimal rate/distance pair summing to one. Our transformation seems to inherently produce an alphabet bit size that is near-linear in $\frac{1}{\varepsilon}$ due to the rate loss stemming from alphabet expansion. For half-errors, ECCs based on algebraic geometry [TV91] achieving alphabet bit size logarithmic in $\frac{1}{\varepsilon}$ are known, but their encoding and decoding complexities are higher. State of the art linear-time expander codes [RT06], which improve over [GI05], have an alphabet bit size which is polylogarithmic in $\frac{1}{\varepsilon}$. Interestingly, we will show in Chapter 4 that, as opposed to half-error ECCs, no such insdel code exist over alphabets that have sub-linear bit-size in terms of $\frac{1}{\varepsilon}$.

3.4 Synchronization Strings

In this section, we formally define and develop ε-synchronization strings, which will be the base index string S in our (n, δ)-indexing solutions. As explained in Section 3.1.2, one can think of the prefixes $S[1, l]$ of an index string S as *codewords* encoding their length l since the prefix $S[1, l]$, or a corrupted version of it, will be exactly all the position-related information that has been received by the time the lth symbol is communicated.

Definition 3.4.1 (Codewords Associated with an Index String). *Given any index string S in a solution to an indexing problem, we define the set of codewords associated with S to be the set of prefixes of S, i.e., $\{S[1, l] \mid 1 \leq l \leq |S|\}$.*

Next, we define a distance metric on any set of strings, which will be useful in quantifying how good an index string S and its associated set of codewords are.

Definition 3.4.2 (Relative Suffix Distance)**.** *For any two strings $S, S' \in \Sigma^*$ we define their relative suffix distance (*RSD*) as follows:*

$$\mathrm{RSD}(S, S') = \max_{k>0} \frac{\mathrm{ED}\Big(S\big(|S|-k, |S|\big], S'\big(|S'|-k, |S'|\big]\Big)}{2k}.$$

Note that this is the normalized edit distance between suffixes of length k in two strings, maximized over k.

Next we show that RSD is indeed a distance which satisfies all properties of a metric for any set of strings. To our knowledge, this metric is new. It is, however, similar in spirit to the suffix distance defined in [BGMO17], which unfortunately is non-symmetric and does not satisfy the triangle inequality but can otherwise be used in a similar manner as RSD in the specific context here (see also Section 3.5.7).

Lemma 3.4.3. *For any strings S_1, S_2, and S_3 we have*

- ***Symmetry:*** $\mathrm{RSD}(S_1, S_2) = \mathrm{RSD}(S_2, S_1)$,
- ***Non-Negativity and Normalization:*** $0 \leq \mathrm{RSD}(S_1, S_2) \leq 1$,
- ***Identity of Indiscernibles:*** $\mathrm{RSD}(S_1, S_2) = 0 \Leftrightarrow S_1 = S_2$, *and*
- ***Triangle Inequality:*** $\mathrm{RSD}(S_1, S_3) \leq \mathrm{RSD}(S_1, S_2) + \mathrm{RSD}(S_2, S_3)$.

In particular, RSD *defines a metric on any set of strings.*

Proof. Symmetry and non-negativity follow directly from the symmetry and non-negativity of edit distance. Normalization follows from the fact that the edit distance between two length k strings can be at most $2k$. To see the identity of indiscernibles note that $\mathrm{RSD}(S_1, S_2) = 0$ if and only if for all k the edit distance of the k-suffix of S_1 and S_2 is zero, i.e., if for every k, the k-suffix of S_1 and S_2 are identical. This is equivalent to S_1 and S_2 being equal. Lastly, the triangle inequality also essentially follows from the triangle inequality for edit distance. To see this let $\delta_1 = \mathrm{RSD}(S_1, S_2)$ and $\delta_2 = \mathrm{RSD}(S_2, S_3)$. By the definition of RSD this implies that for all k the k-suffixes of S_1 and S_2 have edit distance at most $2\delta_1 k$ and the k-suffixes of S_2 and S_3 have edit distance at most $2\delta_2 k$. By the triangle inequality for edit distance, this implies that for every k the k-suffixes of S_1 and S_3 have edit distance at most $(\delta_1 + \delta_2) \cdot 2k$ which implies that $\mathrm{RSD}(S_1, S_3) \leq \delta_1 + \delta_2$. □

With these definitions in place, it remains to find index strings which induce a set of codewords, i.e., prefixes, with large relative suffix distance. It is easy to see that the relative suffix distance for any two strings ending on a different symbol is one. This makes the trivial index string, which uses each symbol in Σ only once, induce an associated set of codewords of optimal minimum-RSD-distance one. Such trivial index strings, however, are not interesting as they require an alphabet size linear in the length of the message. To find good index strings over constant size alphabets, we give the following important definition of an ε-synchronization string. The parameter $0 < \varepsilon < 1$ should be thought of measuring how far a string is from the perfect index string, i.e., a string of n distinct symbols.

Definition 3.4.4 (ε-Synchronization String)**.** *String $S \in \Sigma^n$ is an ε-synchronization string if for every $1 \leq i < j < k \leq n+1$ we have that* $\mathrm{ED}\left(S[i,j), S[j,k)\right) > (1-\varepsilon)(k-i)$.

The next lemma shows that the ε-synchronization string property is strong enough to imply a good minimum relative suffix distance between any two codewords associated with it.

Lemma 3.4.5. *If S is an ε-synchronization string, then* $\mathrm{RSD}(S[1,i], S[1,j]) > 1-\varepsilon$ *for any $i < j$, i.e., any two codewords associated with S have relative suffix distance of at least $1-\varepsilon$.*

Proof. Let $k = j - i$. The ε-synchronization string property of S guarantees that

$$\mathrm{ED}\left(S[i-k, i), S[i,j)\right) > (1-\varepsilon)2k.$$

Note that this holds even if $i - k < 1$. To finish the proof, we point out that the maximization in the definition of RSD includes the term $\frac{\mathrm{ED}(S[i-k,i),S[i,j))}{2k} > 1-\varepsilon$, which implies that $\mathrm{RSD}(S[1,i], S[1,j]) > 1-\varepsilon$. □

3.4.1 Existence and Construction

The next important step is to show that the ε-synchronization strings we just defined exist, particularly, over alphabets whose size is independent of string length n. We show the existence of ε-synchronization strings of arbitrary length for any $\varepsilon > 0$ using an alphabet size which is only polynomially large in $1/\varepsilon$. We remark that ε-synchronization strings can be seen as a strong generalization of square-free sequences in which any two neighboring substrings $S[i,j)$ and $S[j,k)$ only have to be different and not also far from each other in edit distance. Thue [Thu12] famously showed the existence of arbitrarily large square-free strings over a ternary alphabet. Thue's methods for constructing such strings, however, turns out to be fundamentally too weak to prove the existence of ε-synchronization strings, for any constant $\varepsilon < 1$.

Our existence proof requires the general Lovász local lemma which we recall here first:

Lemma 3.4.6 (General Lovász Local Lemma)**.** *Let $\{A_1, \ldots, A_n\}$ be a set of "bad" events. The directed graph $G(V,E)$ is called a dependency graph for this set of events if $V = \{1, \ldots, n\}$ and each event A_i is mutually independent of all the events $\{A_j : (i,j) \notin E\}$. Now, if there exists $x_1, \ldots, x_n \in [0,1)$ such that for all i we have*

$$\mathbb{P}\left[A_i\right] \leq x_i \prod_{(i,j)\in E} (1-x_j)$$

then there exists a way to avoid all events A_i simultaneously and the probability for this to happen is bounded below by

$$\mathbb{P}\left[\bigwedge_{i=1}^{n} \bar{A}_i\right] \geq \prod_{i=1}^{n} (1-x_i) > 0.$$

Theorem 3.4.7. *For any $\varepsilon \in (0,1)$ and $n \geq 1$, there exists an ε-synchronization string of length n over an alphabet of size $\Theta(1/\varepsilon^4)$.*

Proof. Let S be a string of length n obtained by symbol-size concatenating of two strings T and R, where T is simply the repetition of $0, \ldots, t-1$ for $t = \Theta\left(\frac{1}{\varepsilon^2}\right)$, and R is a uniformly random string of length n over alphabet Σ. In other words, $S_i = (i \bmod t, R_i)$. We prove that S is an ε-synchronization string with positive probability by showing that there is a positive probability that S contains no *bad triple*, where (x,y,z) is a bad triple if $\mathrm{ED}(S[x,y), S[y,z)) \leq (1-\varepsilon)(z-x)$.

First, note that a triple (x,y,z) for which $z - x \leq t$ cannot be a bad triple as it consists of completely distinct symbols by courtesy of T. Therefore, it suffices to show that there is no bad triple (x,y,z) in R for x, y, z such that $z - x > t$. Let (x,y,z) be a bad triple and let $a_1 a_2 \cdots a_k$ be the longest common subsequence of $R[x,y)$ and $R[y,z)$. It is straightforward to see that $\mathrm{ED}(R[x,y), R[y,z)) = (y-x) + (z-y) - 2k = z - x - 2k$. Since (x,y,z) is a bad triple, we have that $z - x - 2k \leq (1-\varepsilon)(z-x)$, which means that $k \geq \frac{\varepsilon}{2}(z-x)$. With this observation in mind, we say that $R[x,z)$ is a *bad interval* if it contains a subsequence $a_1 a_2 \cdots a_k a_1 a_2 \cdots a_k$ such that $k \geq \frac{\varepsilon}{2}(z-x)$.

To prove the theorem, it suffices to show that a randomly generated string does not contain any bad intervals with a non-zero probability. We first bound above the probability of an interval of length l being bad.

$$\begin{aligned}
\Pr_{I \sim \Sigma^l}[I \text{ is bad}] &\leq \binom{l}{\varepsilon l} |\Sigma|^{-\frac{\varepsilon l}{2}} \\
&\leq \left(\frac{el}{\varepsilon l}\right)^{\varepsilon l} |\Sigma|^{-\frac{\varepsilon l}{2}} \\
&= \left(\frac{e}{\varepsilon \sqrt{|\Sigma|}}\right)^{\varepsilon l},
\end{aligned}$$

where the first inequality holds because, if an interval of length l is bad, then it must contain a repeating subsequence of length $\frac{l\varepsilon}{2}$. Any such sequence can be specified via εl positions in the l long interval and the probability that a given fixed sequence is repeating for a random string is $|\Sigma|^{-\frac{\varepsilon l}{2}}$. The second inequality comes from the fact that $\binom{n}{k} < \left(\frac{ne}{k}\right)^k$. The resulting inequality shows that the probability of an interval of length l being bad is bounded above by $C^{-\varepsilon l}$, where C can be made arbitrarily large by taking a sufficiently large alphabet size $|\Sigma|$.

To show that there is a non-zero probability that the uniformly random string R contains no bad interval I of size t or larger, we use the general Lovász local lemma stated in Lemma 3.4.6. Note that the badness of interval I is mutually independent of the badness of all intervals that do not intersect I. We need to find real numbers $x_{p,q} \in [0,1)$ corresponding to intervals $R[p,q)$ for which

$$Pr\left[\text{Interval } R[p,q) \text{ is bad}\right] \leq x_{p,q} \prod_{R[p,q) \cap R[p',q') \neq \emptyset} (1 - x_{p',q'}).$$

We have seen that the left-hand side can be upper bounded by $C^{-\varepsilon|R[p,q)|} = C^{\varepsilon(p-q)}$. Furthermore, any interval of length l' intersects at most $l + l'$ intervals of length l. We propose $x_{p,q} = D^{-\varepsilon|R[p,q)|} = D^{\varepsilon(p-q)}$ for some constant $D > 1$. Therefore, it suffices to find a constant D that for all substrings $R[p, q)$ satisfies

$$C^{\varepsilon(p-q)} \leq D^{\varepsilon(p-q)} \prod_{l=t}^{n} \left(1 - D^{-\varepsilon l}\right)^{l+(q-p)},$$

or more clearly, for all $l' \in \{1, \cdots, n\}$ satisfies

$$C^{-l'} \leq D^{-l'} \prod_{l=t}^{n} \left(1 - D^{-\varepsilon l}\right)^{\frac{l+l'}{\varepsilon}},$$

which means that

$$C \geq \frac{D}{\prod_{l=t}^{n} \left(1 - D^{-\varepsilon l}\right)^{\frac{1+l/l'}{\varepsilon}}}. \tag{3.1}$$

For $D > 1$, the right-hand side of Equation (3.1) is maximized when $n = \infty$ and $l' = 1$, and since we want Equation (3.1) to hold for all n and all $l' \in \{1, \cdots, n\}$, it suffices to find a D such that

$$C \geq \frac{D}{\prod_{l=t}^{\infty} \left(1 - D^{-\varepsilon l}\right)^{\frac{l+1}{\varepsilon}}}.$$

To this end, let

$$L = \min_{D>1} \left\{ \frac{D}{\prod_{l=t}^{\infty} \left(1 - D^{-\varepsilon l}\right)^{\frac{l+1}{\varepsilon}}} \right\}.$$

Then, it suffices to have $|\Sigma|$ large enough so that

$$C = \frac{\varepsilon\sqrt{|\Sigma|}}{e} \geq L,$$

which means that $|\Sigma| \geq \frac{e^2 L^2}{\varepsilon^2}$ allows us to use the Lovász local lemma. We claim that $L = \Theta(1)$, which will complete the proof. Since $t = \omega\left(\frac{\log \frac{1}{\varepsilon}}{\varepsilon}\right)$,

$$\forall l \geq t \qquad D^{-\varepsilon l} \cdot \frac{l+1}{\varepsilon} \ll 1.$$

Therefore, we can use the fact that $(1 - x)^k > 1 - xk$ to show the following.

$$\begin{aligned}
\frac{D}{\prod_{l=t}^{\infty} \left(1 - D^{-\varepsilon l}\right)^{\frac{l+1}{\varepsilon}}} &< \frac{D}{\prod_{l=t}^{\infty} \left(1 - \frac{l+1}{\varepsilon} \cdot D^{-\varepsilon l}\right)} && (3.2)\\
&< \frac{D}{1 - \sum_{l=t}^{\infty} \frac{l+1}{\varepsilon} \cdot D^{-\varepsilon l}} && (3.3)\\
&= \frac{D}{1 - \frac{1}{\varepsilon} \sum_{l=t}^{\infty} (l+1) \cdot \left(D^{-\varepsilon}\right)^l} && (3.4)\\
&= \frac{D}{1 - \frac{1}{\varepsilon} \frac{2t(D^{-\varepsilon})^t}{(1-D^{-\varepsilon})^2}} && (3.5)
\end{aligned}$$

$$= \frac{D}{1 - \frac{2}{\varepsilon^3}\frac{D^{-\frac{1}{\varepsilon}}}{(1-D^{-\varepsilon})^2}}. \tag{3.6}$$

Equation (3.3) is derived using the fact that $\prod_{i=1}^{\infty}(1-x_i) \geq 1 - \sum_{i=1}^{\infty} x_i$ and Equation (3.5) is a result of the following equality for $x < 1$:

$$\sum_{l=t}^{\infty}(l+1)x^l = \frac{x^t(1+t-tx)}{(1-x)^2} < \frac{2tx^t}{(1-x)^2}.$$

One can see that for $D = 7$, $\max_{\varepsilon}\left\{\frac{2}{\varepsilon^3}\frac{D^{-\frac{1}{\varepsilon}}}{(1-D^{-\varepsilon})^2}\right\} < 0.9$, and therefore step (3.3) is legal and (3.6) can be upper-bounded by a constant. Hence, $L = \Theta(1)$ and the proof is complete. □

Remarks on the alphabet size: Theorem 3.4.7 shows that for any $\varepsilon > 0$ there exists an ε-synchronization string over alphabets of size $O(\varepsilon^{-4})$. A polynomial dependence on ε is also necessary. In particular, there do not exist any ε-synchronization string over alphabets of size smaller than ε^{-1}. In fact, any $\frac{1}{\varepsilon}$-long substring of an ε-synchronization string has to contain completely distinct elements. This can be easily proven as follows: For sake of contradiction let $S[i, i+\varepsilon^{-1})$ be a substring of an ε-synchronization string where $S[j] = S[j']$ for $i \leq j < j' < i + \varepsilon^{-1}$. Then, $\mathrm{ED}\left(S[j], S[j+1, j'+1))\right) = j' - j - 1 = (j' + 1 - j) - 2 \leq (j' + 1 - j)(1 - 2\varepsilon)$. In Chapter 10, we use the Lovász local lemma together with a more sophisticated non-uniform probability space, which avoids any repeated symbols within a small distance, allows avoiding the use of the string T in our proof and improving the alphabet size to $O(\varepsilon^{-2})$. It seems much harder to improve the alphabet size to $o(\varepsilon^{-2})$ and we are not convinced that it is possible. This work thus leaves open the interesting question of closing the quadratic gap between $O(\varepsilon^{-2})$ and $\Omega(\varepsilon^{-1})$ from either side.

Theorem 3.4.7 also implies an efficient randomized construction.

Lemma 3.4.8. *There exists a randomized algorithm which for any $\varepsilon > 0$ and n constructs an ε-synchronization string of length n over an alphabet of size $O(\varepsilon^{-4})$ with expected running time $O(n^5)$.*

Proof. Using the algorithmic framework for the Lovász local lemma given by Moser and Tardos [MT10] and the extensions by Haeupler et al. [HSS11], one can get such a randomized algorithm from the proof in Theorem 3.4.7. The algorithm starts with a random string over any alphabet Σ of size $C\varepsilon^{-4}$ for some sufficiently large C. It then checks all $O(n^2)$ intervals for a violation of the ε-synchronization string property. For every interval this is an edit distance computation which can be done in $O(n^2)$ time using the classic Wagner-Fischer dynamic programming algorithm. If a violating interval is found the symbols in this interval are assigned fresh random values. This is repeated until no more violations are found. It is shown in [HSS11] that this algorithm performs only $O(n)$ expected number of resamplings. This gives an expected running time of $O(n^5)$ overall, as claimed. □

Lastly, since synchronization strings can be repositioned in a streaming fashion, they can be used in many important applications where the length of the required synchronization string is not known in advance. (see Chapters 7 and 8) In such a setting it is advantageous to have an infinite synchronization string over a fixed alphabet. In particular, since every substring of an ε-synchronization string is also an ε-synchronization string by definition, having an infinite ε-synchronization string also implies the existence for every length n, i.e., Theorem 3.4.7. Interestingly, a simple argument shows that the converse is true as well, i.e., the existence of an ε-synchronization string for every length n implies the existence of an infinite ε-synchronization string over the same alphabet.

Lemma 3.4.9. *For any $\varepsilon \in (0,1)$ there exists an infinite ε-synchronization string over an alphabet of size $\Theta(1/\varepsilon^4)$.*

Proof. Fix any $\varepsilon \in (0,1)$. According to Theorem 3.4.7 there exist an alphabet Σ of size $O(1/\varepsilon^4)$ such that there exists at least one ε-synchronization strings over Σ for every length $n \in \mathbb{N}$. We will define an infinite synchronization string $S = s_1 \cdot s_2 \cdot s_3 \ldots$ with $s_i \in \Sigma$ for any $i \in \mathbb{N}$ for which the ε-synchronization property holds for any neighboring substrings. We define this string inductively. In particular, we fix an ordering on Σ and define $s_1 \in \Sigma$ to be the first symbol in this ordering such that an infinite number of ε-synchronization strings over Σ starts with s_1. Given that there is an infinite number of ε-synchronization over Σ such an s_1 exists. Furthermore, the subset of ε-synchronization strings over Σ which start with s_1 is infinite by definition, allowing us to define $s_2 \in \Sigma$ to be the lexicographically first symbol in Σ such there exists an infinite number of ε-synchronization strings over Σ starting with $s_1 \cdot s_2$. In the same manner, we inductively define s_i to be the lexicographically first symbol in Σ for which there exists and infinite number of ε-synchronization strings over Σ starting with $s_1 \cdot s_2 \cdot \ldots \cdot s_i$. To see that the infinite string defined in this manner does indeed satisfy the edit distance requirement of the ε-synchronization property defined in Definition 3.4.4, we note that for every $i < j < k$ with $i, j, k \in \mathbb{N}$ there exists, by definition, an ε-synchronization string, and in fact an infinite number of them, which contains $S[1,k]$ and thus also $S[i,k]$ as a substring implying that indeed $\mathrm{ED}\left(S[i,j), S[j,k)\right) > (1-\varepsilon)(k-i)$ as required. Our definition thus produces the unique lexicographically first infinite ε-synchronization string over Σ. □

We remark that any string produced by the randomized construction of Lemma 3.4.8 is guaranteed to be a correct ε-synchronization string and the only randomized aspect of the algorithm is its running time. This randomized synchronization string construction is furthermore only needed once as a pre-processing step. The encoder or decoder of any resulting error correcting codes do not require any randomization. Furthermore, in Section 3.5 we will provide a deterministic polynomial time construction of a relaxed version of ε-synchronization strings that can still be used as a basis for good (n, δ)-indexing algorithms thus leading to insdel codes with a deterministic polynomial time code construction as well.

It nonetheless remains interesting to obtain fast deterministic constructions of finite and infinite ε-synchronization strings. In Chapter 8, we achieve such efficient deterministic constructions for ε-synchronization strings. These constructions even produce the infinite

ε-synchronization string S proven to exist by Lemma 3.4.9, which is much less explicit: While for any n and ε an ε-synchronization string of length n can in principle be found using an exponential time enumeration, there is no straightforward algorithm which follows the proof of Lemma 3.4.9 and given an $i \in \mathbb{N}$ produces the ith symbol of such an S in a finite amount of time (bounded by some function in i). Our constructions in Chapter 8 require significantly more work but in the end lead to an explicit deterministic construction of an infinite ε-synchronization string for any $\varepsilon > 0$ for which the ith symbol can be computed in only $O(\log i)$ time – thus satisfying one of the strongest notions of constructiveness that can be achieved.

3.4.2 Repositioning Algorithm for ε-Synchronization Strings

We now provide an algorithm for repositioning synchronization strings, i.e., an algorithm that can form a solution to the indexing problem along with ε-synchronization strings. In the beginning of Section 3.4, we introduced the notion of relative suffix distance between two strings. Theorem 3.4.5 stated a lower bound of $1-\varepsilon$ for relative suffix distance between any two distinct codewords associated with an ε-synchronization string, i.e., its prefixes. Hence, a natural repositioning scheme for guessing the position of a received symbol would be finding the prefix of the index string with the closest relative suffix distance to the string received thus far. We call this algorithm *the minimum relative suffix distance decoding algorithm.*

We define the notion of *relative suffix error density at index symbol i* which represents the maximized density of errors taken place over suffixes of $S[1, i]$. We will show that this decoding procedure works correctly as long as the relative suffix error density is not larger than $\frac{1-\varepsilon}{2}$. Then, we will show that if adversary is allowed to perform c insertions or deletions, the relative suffix distance may exceed $\frac{1-\varepsilon}{2}$ upon arrival of at most $\frac{2c}{1-\varepsilon}$ many successfully transmitted symbols. Finally, we will deduce that this repositioning scheme correctly decodes all but $\frac{2c}{1-\varepsilon}$ many index symbols that are successfully transmitted. Formally, we claim that:

Theorem 3.4.10. *Any ε-synchronization string of length n along with the minimum relative suffix distance decoding algorithm form a solution to (n, δ)-indexing problem that guarantees $\frac{2}{1-\varepsilon}n\delta$ or less misdecodings. This repositioning algorithm is streaming and can be implemented in a way that works in $O(n^4)$ time.*

Before proceeding to the proof of the claim above, we first provide the following useful definitions.

Definition 3.4.11 (Error Count Function)**.** *Let S be an index string sent over an insertion-deletion channel. We denote the* error count from index i to index j *with $\mathcal{E}(i, j)$ and define it to be the number of insdels applied to S from the moment $S[i]$ is sent until the moment $S[j]$ is sent. $\mathcal{E}(i, j)$ would count the deletion of $S[j]$, however, it would* not *count the deletion of $S[i]$.*

Definition 3.4.12 (Relative Suffix Error Density)**.** *Let string S be sent over an insertion-deletion channel and let $\mathcal{E}$ denote the corresponding error count function. We define the*

relative suffix error density *of the communication as*

$$\max_{i\geq 1} \frac{\mathcal{E}\left(|S|-i,|S|\right)}{i}.$$

The following lemma relates the suffix distance of the string sent by the sender and the one received by the receiver at any point of a communication over an insertion-deletion channel to the relative suffix error density of the communication at that point.

Lemma 3.4.13. *Let string S be sent over an insertion-deletion channel and the corrupted version of it S' be received on the other end. The relative suffix distance between S and S', $\mathrm{RSD}(S,S')$, is at most the relative suffix error density of the communication.*

Proof. Let $\tilde{\tau}=(\tilde{\tau}_1,\tilde{\tau}_2)$ be the string matching from S to S' that characterizes insdels that have turned S into S'. Then:

$$\begin{aligned}
\mathrm{RSD}(S,S') &= \max_{k>0} \frac{\mathrm{ED}(S(|S|-k,|S|],S'(|S'|-k,|S'|])}{2k} && (3.7)\\
&= \max_{k>0} \frac{\min_{\tau:S(|S|-k,|S|]\to S'(|S'|-k,|S'|]}\{sc(\tau_1)+sc(\tau_2)\}}{2k} && (3.8)\\
&\leq \max_{k>0} \frac{2(sc(\tilde{\tau}_1^k)+sc(\tilde{\tau}_2^k))}{2k} \leq \text{Relative Suffix Error Density} && (3.9)
\end{aligned}$$

where $\tilde{\tau}^k$ is $\tilde{\tau}$ limited to its suffix corresponding to $S(|S|-k,|S|])$. Note that (3.7) and (3.8) follow from the definitions of edit distance and relative suffix distance. Moreover, to verify step (3.9), one has to note that one single insertion or deletion on the k-element suffix of a string may result in a string with k-element suffix of edit distance two of the original string's k-element suffix; one stemming from the inserted/deleted symbol and the other one stemming from a symbol appearing/disappearing at the beginning of the suffix in order to keep the size of suffix k. □

A key consequence of Lemma 3.4.13 is that if an ε-synchronization string is being sent over an insertion-deletion channel and at some step the relative suffix error density corresponding to errors is smaller than $\frac{1-\varepsilon}{2}$, the relative suffix distance of the sent string and the received one at that point is smaller than $\frac{1-\varepsilon}{2}$; therefore, as RSD of all pairs of codewords associated with an ε-synchronization string are greater than $1-\varepsilon$, the receiver can correctly decode the index of–or guess the position of–the last received symbol of the communication by simply finding the codeword (i.e., prefix of the synchronization string) with minimum relative suffix distance to the string received so far.

The following lemma states that such a guarantee holds most of the time during the transmission of a synchronization string.

Lemma 3.4.14. *Let ε-synchronization string S be sent over an insertion-channel channel and corrupted string S' be received on the other end. If there are c_i symbols inserted and c_d symbols deleted, then, for any integer t, the relative suffix error density is smaller than $\frac{1-\varepsilon}{t}$ upon arrival of all but $\frac{t(c_i+c_d)}{1-\varepsilon}-c_d$ many of the successfully transmitted symbols.*

Proof. Let $\mathcal{E}$ denote the error count function of the communication. We define the potential function Φ over $\{0, 1, \cdots, n\}$ as follows.

$$\Phi(i) = \max_{1 \le s \le i} \left\{ \frac{t \cdot \mathcal{E}(i-s, i)}{1-\varepsilon} - s \right\}$$

Also, set $\Phi(0) = 0$. We prove the theorem by showing the correctness of the following claims:

1. If $\mathcal{E}(i-1, i) = 0$, i.e., the adversary does not insert or delete any symbols in the interval starting right after the moment $S[i-1]$ is sent and ending at when $S[i]$ is sent, then the value of Φ drops by 1 or becomes/stays zero, i.e., $\Phi(i) = \max\{0, \Phi(i-1) - 1\}$.

2. If $\mathcal{E}(i-1, i) = k$, i.e., adversary inserts or deletes k symbols in the interval starting right after the moment $S[i-1]$ is sent and ending at when $S[i]$ is sent, then the value of Φ increases by $\frac{tk}{1-\varepsilon} - 1$, i.e., $\Phi(i) = \Phi(i-1) + \frac{tk}{1-\varepsilon} - 1$.

3. If $\Phi(i) = 0$, then the relative suffix error density of the string that is received when $S[i]$ arrives at the receiving side is not larger than $\frac{1-\varepsilon}{t}$.

Given the correctness of claims made above, the lemma can be proved as follows. As adversary can apply at most $c_i + c_d$ insertions or deletions, Φ can gain a total increase of $\frac{t \cdot (c_i + c_d)}{1-\varepsilon}$ minus the number of is for which $\Phi(i) = 0$ but $\Phi(i+1) \neq 0$. Since Φ always drops by one or to zero and increases non-integrally, the value of Φ can be non-zero for at most $\frac{t \cdot (c_i + c_d)}{1-\varepsilon}$ many inputs. As the value of $\Phi(i)$ is non-zero for all i's where $S[i]$ has been removed by the adversary, there are at most $\frac{t \cdot (c_i + c_d)}{1-\varepsilon} - c_d$ elements i where $\Phi(i)$ is non-zero and i is successfully transmitted. Hence, at most $\frac{t \cdot (c_i + c_d)}{1-\varepsilon} - c_d$ many of correctly transmitted symbols can possibly be decoded incorrectly.

We now proceed to the proof of each of the above-mentioned claims to finish the proof:

1. In this case, $\mathcal{E}(i-s, i) = \mathcal{E}(i-s, i-1)$. So,

$$\begin{aligned}
\Phi(i) &= \max_{1 \le s \le i} \left\{ \frac{t \cdot \mathcal{E}(i-s, i)}{1-\varepsilon} - s \right\} \\
&= \max_{1 \le s \le i} \left\{ \frac{t \cdot \mathcal{E}(i-s, i-1)}{1-\varepsilon} - s \right\} \\
&= \max \left\{ 0, \max_{2 \le s \le i} \left\{ \frac{t \cdot \mathcal{E}(i-s, i-1)}{1-\varepsilon} - s \right\} \right\} \\
&= \max \left\{ 0, \max_{1 \le s \le i-1} \left\{ \frac{t \cdot \mathcal{E}(i-1-s, i-1)}{1-\varepsilon} - s - 1 \right\} \right\} \\
&= \max\{0, \Phi(i-1) - 1\}
\end{aligned}$$

2. In this case, $\mathcal{E}(i-s,i) = \mathcal{E}(i-s,i-1)+k$. So,

$$\begin{aligned}
\Phi(i) &= \max_{1\le s\le i}\left\{\frac{t\cdot\mathcal{E}(i-s,i)}{1-\varepsilon}-s\right\}\\
&= \max\left\{\frac{tk}{1-\varepsilon}-1, \max_{2\le s\le i}\left\{\frac{t\cdot\mathcal{E}(i-s,i-1)+tk}{1-\varepsilon}-s\right\}\right\}\\
&= \max\left\{\frac{tk}{1-\varepsilon}-1, \frac{tk}{1-\varepsilon}+\max_{1\le s\le i-1}\left\{\frac{t\cdot\mathcal{E}(i-1-s,i-1)}{1-\varepsilon}-s-1\right\}\right\}\\
&= \frac{tk}{1-\varepsilon}-1+\max\left\{0, \max_{1\le s\le i-1}\left\{\frac{t\cdot\mathcal{E}(i-1-s,i-1)}{1-\varepsilon}-s\right\}\right\}\\
&= \frac{tk}{1-\varepsilon}-1+\max\left\{0,\Phi(i-1)\right\}\\
&= \Phi(i-1)+\frac{tk}{1-\varepsilon}-1
\end{aligned}$$

3. And finally,

$$\begin{aligned}
&\Phi(i) = \max_{1\le s\le i}\left\{\frac{t\cdot\mathcal{E}(i-s,i)}{1-\varepsilon}-s\right\} = 0\\
\Rightarrow\ &\forall 1\le s\le i:\ \frac{t\cdot\mathcal{E}(i-s,i)}{1-\varepsilon}-s\le 0\\
\Rightarrow\ &\forall 1\le s\le i:\ t\cdot\mathcal{E}(i-s,i)\le s(1-\varepsilon)\\
\Rightarrow\ &\forall 1\le s\le i:\ \frac{\mathcal{E}(i-s,i)}{s}\le\frac{1-\varepsilon}{t}\\
\Rightarrow\ &\text{Relative Suffix Error Density } = \max_{1\le s\le i}\left\{\frac{\mathcal{E}(i-s,i)}{s}\right\}\le\frac{1-\varepsilon}{t}
\end{aligned}$$

These finish the proof of the lemma. □

Now, we have all necessary tools to analyze the performance of the minimum relative suffix distance decoding algorithm:

Proof of Theorem 3.4.10. As adversary is allowed to insert or delete up $n\delta$ symbols, by Lemma 3.4.14, there are at most $\frac{2n\delta}{1-\varepsilon}$ successfully transmitted symbols during the arrival of which at the receiving side, the relative suffix error density is greater than $\frac{1-\varepsilon}{2}$; Hence, by Lemma 3.4.13, there are at most $\frac{2n\delta}{1-\varepsilon}$ misdecoded successfully transmitted symbols.

Further, we remark that this algorithm can be implemented in $O(n^4)$ time. Using dynamic programming, we can pre-process the edit distance of any substring of S, like $S[i,j]$ to any substring of S', like $S'[i',j']$, in $O(n^4)$ time. Then, to decode each index symbol like $S'[l']$ (i.e., guess its original position), we can find the codeword with minimum relative suffix distance to $S'[1,l']$ by calculating the relative suffix distance of it to all n codewords. Finding suffix distance of $S'[1,l']$ and a codeword like $S[1,l]$ can also be simply done by minimizing $\frac{\mathrm{ED}(S(l-k,l],S'(l'-k,l'])}{k}$ over all k. This can be done in $O(n)$ time. With an

$O(n^4)$ pre-process and an $O(n^3)$ computation as mentioned above, we have shown that the decoding process can be implemented in $O(n^4)$ time. Finally, this algorithm clearly satisfies streaming property as it decodes indices of arrived symbols merely using the symbols which have arrived earlier. □

We remark that by taking $\varepsilon = o(1)$, one can obtain a solution to the (n, δ)-indexing problem with a misdecoding guarantee of $2n\delta(1+o(1))$ which, using Theorem 3.3.1, results into a translation of $n\delta$ insertions and deletions into $n\delta(5 + o(1))$ half-errors.

In Section 3.5.7 , we show that this guarantee of the min-distance-decoder can be slightly improved to $n\delta(3 + o(1))$ half-errors, at the cost of some simplicity. In particular, one can go beyond an RSD distance of $\frac{1-\varepsilon}{2}$ by considering the relative suffix pseudo distance RSPD, which was introduced in [BGMO17], as an alternative distance measure. RSPD can act as a stand-in metric for the minimum-distance decoder and lead to the above-mentioned slightly improved decoding guarantees, despite neither being symmetric nor satisfying the triangle inequality. More precisely, for any set of $k = k_i + k_d$ insdel errors consisting of k_i insertions and k_d deletions the RSPD based indexing solution leads to at most $(1 + \varepsilon)(3k_i + k_d)$ half-errors which does imply "near-MDS" codes for deletion-only channels but still falls short for general insdel errors.

This leaves open the intriguing question whether a further improved (pseudo) distance definition can achieve an indexing solution with negligible number of misdecodings for the minimum-distance decoder.

3.5 More Advanced Repositioning Algorithms and ε-Self Matching Property

Thus far, we have introduced ε-synchronization strings as fitting solutions to the indexing problem. In Section 3.4.2, we provided an algorithm to solve the indexing problem along with synchronization strings with an asymptotic guarantee of $2n\delta$ misdecodings. As explained in Section 3.1.3, such a guarantee falls short of giving Theorem 3.1.1. In this section, we thus provide a variety of more advanced repositioning algorithms that provide better decoding guarantees, in particular, achieve a misdecoding fraction which goes to zero as ε goes to zero.

We start by pointing out a very useful property of ε-synchronization strings in Section 3.5.1. We define a monotone matching between two strings as a common subsequence of them. We will next show that in a monotone matching between an ε-synchronization string and itself, the number of matches that both correspond to the same element of the string is fairly large. We will refer to this property as ε-self-matching property. We show that one can very formally think of this ε-self-matching property as a robust global guarantee in contrast to the factor-closed strong local requirements of the ε-synchronization property. One advantage of this relaxed notion of ε-self-matching is that one can show that a random string over alphabets polynomially large in ε^{-1} satisfies this property (Section 3.5.2). This leads to a particularly simple generation process for S. Finally, showing

that this property even holds for approximately $\log n$-wise independent strings directly leads to a deterministic polynomial time algorithm generating such strings as well.

In Section 3.5.3, we propose a repositioning algorithm for insdel errors that basically works by finding monotone matchings between the received string and the synchronization string. Using the ε-self-matching property we show that this algorithm guarantees $O\left(n\sqrt{\varepsilon}\right)$ misdecodings. This algorithm works in $O(n^2/\sqrt{\varepsilon})$ time and is exactly what we need to prove our main theorem.

In Sections 3.5.4 and 3.5.5 we provide two simpler linear-time repositioning algorithms that solve the indexing problem under the assumptions that the adversary can only delete symbols or only insert symbols. These algorithms not only guarantee asymptotically optimal $\frac{\varepsilon}{1-\varepsilon}n\delta$ misdecodings but are also error-free. In Section 3.5.6, we present linear-time near-MDS insertion-only and deletion-only codes that can be derived by these repositioning algorithms.

Finally, in Section 3.5.7, we present an improved version of the minimum RSD decoding algorithm that achieves a better misdecoding guarantee by replacing RSD with a similar pseudo-distance.

See Table 3.1 for a break down of the repositioning schemes presented in this chapter, the type of error under which they work, the number of misdecodings they guarantee, whether they are error-free or streaming, and their repositioning time complexities.

3.5.1 ε-Self Matching Property

Before proceeding to the main results of this section, we define *monotone matchings* as follows.

Definition 3.5.1 (Monotone Matchings)**.** *A monotone matching between S and S' is a set of pairs of indexes like:*

$$M = \{(a_1, b_1), \cdots, (a_m, b_m)\}$$

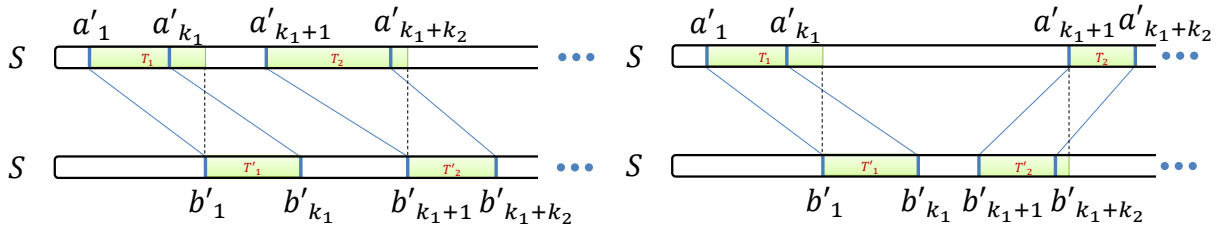
where $a_1 < \cdots < a_m$, $b_1 < \cdots < b_m$, and $S[a_i] = S'[b_i]$.

We now point out a key property of synchronization strings that will be broadly used in our repositioning algorithms in Theorem 3.5.2 which, basically, states that two large similar subsequences of an ε-synchronization string cannot disagree on many positions. More formally, let $M = \{(a_1, b_1), \cdots, (a_m, b_m)\}$ be a monotone matching between S and itself. We call the pair (a_i, b_i) a *good pair* if $a_i = b_i$ and a *bad pair* otherwise. Then:

Theorem 3.5.2. *Let S be an ε-synchronization string of size n and $M = \{(a_1, b_1), \cdots, (a_m, b_m)\}$ be a monotone matching of size m from S to itself containing g good pairs and b bad pairs. Then,*

$$b < \varepsilon(n - g)$$

Proof. Let $(a'_1, b'_1), \cdots, (a'_b, b'_b)$ indicate the set of bad pairs in M indexed as $a'_1 < \cdots < a'_b$ and $b'_1 < \cdots < b'_b$. Without loss of generality, assume that $a'_1 < b'_1$. Let k_1 be the largest integer such that $a'_{k_1} < b'_1$. Then, the pairs $(a'_1, b'_1), \cdots, (a'_{k_1}, b'_{k_1})$ form a common



(a) $a'_{k_1+1} < b'_{k_1+1}$ (b) $a'_{k_1+1} > b'_{k_1+1}$

Figure 3.1: Pictorial representation of T_2 and T'_2

subsequence of size k_1 between $T_1 = S[a'_1, b'_1)$ and $T'_1 = S[b'_1, b'_{k_1}]$. Now, the synchronization string guarantee implies that:

$$\begin{aligned} k_1 &\leq \mathrm{LCS}(T_1, T'_1) \\ &< \frac{|T_1| + |T'_1| - \mathrm{ED}\,(T_1, T'_1)}{2} \\ &\leq \frac{\varepsilon(|T_1| + |T'_1|)}{2} \end{aligned}$$

Note that the monotonicity of the matching guarantees that there are no good matches occurring on indices covered by T_1 and T'_1, i.e., $a'_1, \cdots, b'_{k_1}$. One can repeat very same argument for the remaining bad matches to rule out bad matches $(a'_{k_1+1}, b'_{k_1+1}), \cdots, (a'_{k_1+k_2}, b'_{k_1+k_2})$ for some k_2 having the following inequality guaranteed:

$$k_2 < \frac{\varepsilon(|T_2| + |T'_2|)}{2} \tag{3.10}$$

where

$$\begin{cases} T_2 = [a'_{k_1+1}, b'_{k_1+1}) \text{ and } T'_2 = [b'_{k_1+1}, b'_{k_1+k_2}] & a'_{k_1+1} < b'_{k_1+1} \\ T_2 = [b'_{k_1+1}, a'_{k_1+1}) \text{ and } T'_2 = [a'_{k_1+1}, a'_{k_1+k_2}] & a'_{k_1+1} > b'_{k_1+1} \end{cases}$$

For a pictorial representation see Figure 3.1.

Continuing the same procedure, one can find $k_1, \cdots, k_l$, $T_1, \cdots, T_l$, and $T'_1, \cdots, T'_l$ for some l. Summing up all inequalities of form (3.10), we will have:

$$\sum_{i=1}^{l} k_i < \frac{\varepsilon}{2} \cdot \left(\sum_{i=1}^{l} |T_i| + \sum_{i=1}^{l} |T'_i| \right) \tag{3.11}$$

Note that $\sum_{i=1}^{l} k_i = b$ and T_is are mutually exclusive and contain good pair. Same holds for T'_is. Hence, $\sum_{i=1}^{l} |T_i| \leq n - g$ and $\sum_{i=1}^{l} |T'_i| \leq n - g$. All these along with (3.11) give that

$$b < \frac{\varepsilon}{2} \cdot 2\,(n - g) = \varepsilon(n - g).$$

□

We define the ε-self-matching property as follows:

Definition 3.5.3 (ε-Self-Matching Property)**.** *String S satisfies ε-self-matching property if any monotone matching between S and itself contains less than $\varepsilon|S|$ bad pairs.*

Note that ε-synchronization property concerns all substrings of a string while the ε-self-matching property only concerns the string itself. Granted that, we now show that ε-synchronization property and satisfying ε-self-matching property on all substrings are equivalent up to a factor of two:

Theorem 3.5.4. *ε-synchronization and ε-self matching properties are related in the following way:*

a) If S is an ε-synchronization string, then all substrings of S satisfy ε-self-matching property.

b) If all substrings of string S satisfy the $\frac{\varepsilon}{2}$-self-matching property, then S is ε-synchronization string.

Proof of Theorem 3.5.4 (a). This part is a straightforward consequence of Theorem 3.5.2. □

Proof of Theorem 3.5.4 (b). Assume by contradiction that there are $i < j < k$ such that $\mathrm{ED}(S[i,j), S[j,k)) \leq (1-\varepsilon)(k-i)$. Then, $\mathrm{LCS}(S[i,j), S[j,k)) \geq \frac{k-i-(1-\varepsilon)(k-i)}{2} = \frac{\varepsilon}{2}(k-i)$. The corresponding pairs of such longest common subsequence form a monotone matching of size $\frac{\varepsilon}{2}(k-i)$ which contradicts $\frac{\varepsilon}{2}$-self-matching property of S. □

The repositioning algorithms we will propose for ε-synchronization strings in Sections 3.5.3, 3.5.4, and 3.5.5 only make use of the ε-self-matching property of ε-synchronization strings. We now define ε-bad elements which will enable us to show that ε-self matching property, as opposed to the ε-synchronization property, is robust against local changes.

Definition 3.5.5 (ε-bad element)**.** *We call element k of string S an ε-bad element if there exists a factor $S[i,j]$ of S with $i \leq k \leq j$ where $S[i,j]$ does not satisfy the ε-self-matching property. In this case, we also say that element k blames interval $[i,j]$.*

Using the notion of ε-bad elements, we now present Lemma 3.5.6. This lemma suggests that a string containing limited fraction of ε-bad elements would still be an ε'-self matching string for some $\varepsilon' > \varepsilon$. An important consequence of this result is that if one changes a limited number of elements in a given ε-self matching string, the self matching property will be essentially preserved to a lesser extent. Note that no such robustness holds for ε-synchronization strings.

Lemma 3.5.6. *If the fraction of ε-bad elements in string S is less than γ, then S satisfies $(\varepsilon + 2\gamma)$-self matching property.*

Proof. Consider a matching from S to itself. The number of bad matches whose both ends refer to non-ε-bad elements of S is at most $|S|\varepsilon$ by the definition of ε-bad elements. Further, each ε-bad element can appear at most once in each end of bad pairs. Therefore, the number of bad pairs in S can be at most:

$$|S|\varepsilon + 2|S|\gamma = |S|(\varepsilon + 2\gamma)$$

which, by definition, implies that S satisfies the $(\varepsilon + 2\gamma)$-self-matching property. □

On the other hand, in the following lemma, we will show that within a given ε-self matching string, there can be a limited number of ε'-bad elements for sufficiently large $\varepsilon' > \varepsilon$.

Lemma 3.5.7. *Let S be an ε-self matching string of length n. Then, for any $3\varepsilon < \varepsilon' < 1$, at most $\frac{3n\varepsilon}{\varepsilon'}$ many elements of S can be ε'-bad.*

Proof. Let $s_1, s_2, \cdots, s_k$ be ε'-bad elements of S and ε'-bad element s_i blame substring $S[a_i, b_i)$. As intervals $S[a_i, b_i)$ are supposed to be bad, there has to be an ε'-self matching within each $S[a_i, b_i)$ like M_i for which $|M_i| \geq \varepsilon' \cdot |[a_i, b_i)|$. We claim that one can choose a subset of $[1..k]$ like I for which

- Intervals that correspond to the indices in I are mutually exclusive. In other words, for any $i, j \in I$ where $i \neq j$, $[a_i, b_i) \cap [a_j, b_j) = \emptyset$.
- $\sum_{i \in I} |[a_i, b_i)| \geq \frac{k}{3}$.

If such I exists, one can take $\bigcup_{i \in I} M_i$ as a self-matching in S whose size is larger than $\frac{k\varepsilon'}{3}$. As S is an ε-self matching string,

$$\frac{k\varepsilon'}{3} \leq n\varepsilon \Rightarrow k \leq \frac{3n\varepsilon}{\varepsilon'}$$

which finishes the proof. The only remaining piece is proving the claim. Note that any element in $\bigcup_i [a_i, b_i)$ is ε'-bad as they, by definition, belong to an interval with an ε'-self matching. Therefore, $|\bigcup_i [a_i, b_i)| = k$. In order to find set I, we greedily choose the largest substring $[a_i, b_i)$, put its corresponding index into I and then remove any interval intersecting $[a_i, b_i)$. We continue repeating this procedure until all substrings are removed. The set I obtained by this procedure clearly satisfies the first claimed property. Moreover, note that if $l_i = |[a_i, b_i)|$, any interval intersecting $[a_i, b_i)$ falls into $[a_i - l_i, b_i + l_i)$ which is an interval of length $3l_i$. This certifies the second property and finishes the proof. □

As the final remark on the ε-self matching property and its relation with the more strict ε-synchronization property, we show that using the minimum RSD decoder together with an ε-self matching string leads to indexing solutions with a guarantee on the misdecoding count which is only slightly weaker than the guarantee obtained by ε-synchronization strings. In order to do so, we first show that the $(1 - \varepsilon)$ RSD distance property of prefixes holds for any non-ε-bad element in any arbitrary string in Theorem 3.5.8. Then, using Theorem 3.5.8 and Lemma 3.5.7, we upper-bound the number of misdecodings that may occur using a minimum RSD decoder along with an ε-self matching string in Theorem 3.5.9.

Lemma 3.5.8. *Let S be an arbitrary string of length n and $1 \leq i \leq n$ be such that ith element of S is not ε-bad. Then, for any $j \neq i$,* $\mathrm{RSD}(S[1,i], S[1,j]) > 1-\varepsilon$.

Proof. Without loss of generality assume that $j < i$. Consider the interval $[2j-i+1, i]$. As i is not ε-bad, there is no self matching of size $2\varepsilon(i-j)$ within $[2j-i, i]$. In particular, the edit distance of $S[2j-i+1, j]$ and $[j+1, i]$ has to be larger than $(1-\varepsilon)\cdot 2(i-j)$ which equivalently means $\mathrm{RSD}(S[1,i], S[1,j]) > 1-\varepsilon$. Note that if $2j-i+1 < 0$ the proof goes through by simply replacing $2j-i+1$ with zero. □

Theorem 3.5.9. *Using any ε-self matching string along with minimum RSD algorithm, one can solve the (n,δ)-indexing problem with no more than $n(4\delta+6\varepsilon)$ misdecodings.*

Proof. Note that applying Lemma 3.5.7 for ε' gives that there are at most $\frac{3n\varepsilon}{\varepsilon'}$ indices in S that are ε'-bad. Further, using Theorem 3.4.10 and Lemma 3.5.8, at most $\frac{2n\delta}{1-\varepsilon'}$ many of the other indices might be decoded incorrectly upon their arrivals. Therefore, this solution for the (n,δ)-indexing problem can contain at most $n\left(\frac{3\varepsilon}{\varepsilon'} + \frac{2\delta}{1-\varepsilon'}\right)$ many incorrectly decoded indices. Setting $\varepsilon' = \frac{3\varepsilon}{3\varepsilon+2\delta}$ gives an upper bound of $n(4\delta+6\varepsilon)$ on the number of misdecodings. □

3.5.2 Efficient Polynomial Time Construction of ε-Self Matching Strings

In this section, we will use Lemma 3.5.6 to show that there is a polynomial deterministic construction of a string of length n with the ε-self-matching property, which leads to a deterministic efficient code construction. We start by showing that even random strings satisfy the ε-selfmatching property if the alphabet size is an adequately large polynomial in terms of ε^{-1}.

Theorem 3.5.10. *A random string over an alphabet of size $O(\varepsilon^{-3})$ satisfies ε-selfmatching property with a constant probability.*

Proof. Let S be a random string on alphabet Σ of size $|\Sigma| = O(\varepsilon^{-3})$ with an adequately large constant factor hidden in the O notation that we determine later. We are going to find the expected number of ε-bad elements in S. We first count the expected number of ε-bad elements that blame intervals of length $\frac{2}{\varepsilon}$ or smaller. If element k blames interval $S[i,j]$ where $j-i < 2\varepsilon^{-1}$, there has to be two identical symbols appearing in $S[i,j]$ which implies that that there are two identical symbols in the $4\varepsilon^{-1}$-long interval $S(k-2\varepsilon^{-1}, k+2\varepsilon^{-1})$. Therefore, the probability of element k being ε-bad blaming $S[i,j]$ for some $j-i < 2\varepsilon^{-1}$ can be upper-bounded by $\binom{4\varepsilon^{-1}}{2}\frac{1}{|\Sigma|} \leq 8\varepsilon$ if $|\Sigma| \geq \varepsilon^{-3}$. Thus, the expected fraction of ε-bad indices that blame intervals of length $\frac{2}{\varepsilon}$ or smaller is less than 8ε.

We now proceed to finding the expected fraction of ε-bad elements in S blaming intervals of length $2\varepsilon^{-1}$ or more. Since every interval of length l which does not satisfy ε-self-matching property causes at most l ε-bad elements, the expected fraction of such

ε-bad elements, i.e., γ', is at most:

$$\begin{aligned}
\mathbb{E}[\gamma'] &= \frac{1}{n}\sum_{l=2\varepsilon^{-1}}^{n}\sum_{i=1}^{n} l \cdot \Pr[S[i,i+l) \text{ does not satisfy } \varepsilon\text{-self-matching property}] \\
&= \sum_{l=2\varepsilon^{-1}}^{n} l \cdot \Pr[S[i,i+l) \text{ does not satisfy } \varepsilon\text{-self-matching property}] \\
&\leq \sum_{l=2\varepsilon^{-1}}^{n} l\binom{l}{l\varepsilon}^2 \frac{1}{|\Sigma|^{l\varepsilon}} \qquad (3.12)
\end{aligned}$$

Last inequality holds because the number of possible matchings is at most $\binom{l}{l\varepsilon}^2$. Further, fixing the matching edges, the probability of the elements corresponding to pair (a,b) of the matching being identical is independent from all pairs (a',b') where $a' < a$ and $b' < b$. Hence, the probability of the set of pairs forming a matching between the random string S and itself is $\frac{1}{|\Sigma|^{l\varepsilon}}$. Then,

$$\begin{aligned}
\mathbb{E}[\gamma'] &\leq \sum_{l=2\varepsilon^{-1}}^{n} l\left(\frac{le}{l\varepsilon}\right)^{2l\varepsilon}\frac{1}{|\Sigma|^{l\varepsilon}} \\
&\leq \sum_{l=2\varepsilon^{-1}}^{n} l\left(\frac{e}{\varepsilon\sqrt{|\Sigma|}}\right)^{2\varepsilon l} \\
&\leq \sum_{l=2\varepsilon^{-1}}^{\infty} l\left[\left(\frac{e}{\varepsilon\sqrt{|\Sigma|}}\right)^{2\varepsilon}\right]^{l}
\end{aligned}$$

Note that series $\sum_{l=2\varepsilon^{-1}}^{\infty} lx^l = \frac{2\varepsilon^{-1}x^{2\varepsilon^{-1}} - (2\varepsilon^{-1}-1)x^{2\varepsilon^{-1}+1}}{(1-x)^2}$ for $|x| < 1$. Therefore, for $0 < x < \frac{1}{2}$, $\sum_{l=l_0}^{\infty} lx^l < 8\varepsilon^{-1}x^{2\varepsilon^{-1}}$. So,

$$\mathbb{E}[\gamma'] \leq 8\varepsilon^{-1}\left(\frac{e}{2\varepsilon\sqrt{|\Sigma|}}\right)^{4\varepsilon\varepsilon^{-1}} = \frac{e^4}{2}\varepsilon^{-5}\frac{1}{|\Sigma|^2} \leq \frac{e^4}{2}\varepsilon$$

Lemma 3.5.6 implies that this random structure has to satisfy $(\varepsilon+2\gamma)$-self-matching property where

$$\mathbb{E}[\varepsilon + 2\gamma] = \varepsilon + 16\varepsilon + e^4\varepsilon = O(\varepsilon)$$

Therefore, using Markov inequality, a randomly generated string over alphabet $O(\varepsilon^{-3})$ satisfies ε-matching property with constant probability. The constant probability can be as high as one wishes by applying higher constant factor in alphabet size. □

As the next step, we prove a similar claim for strings of length n whose symbols are chosen from an $n^{-O(1)}$-approximate $\Theta\left(\frac{\log n}{\log(1/\varepsilon)}\right)$-wise independent [NN93] distribution over a larger, yet still $\varepsilon^{-O(1)}$ size, alphabet. This is the key step in allowing for a derandomization

using the small sample spaces of Naor and Naor [NN93]. The proof of Theorem 3.5.11 follows a similar strategy as was used in [CGH13] to derandomize the constructive Lovász local lemma. In particular, the crucial idea that is stated in Lemma 3.5.12, is to show that for any large obstruction there has to exist a smaller yet not too small obstruction. This allows one to prove that in the absence of any small and medium size obstructions no large obstructions exist either.

Theorem 3.5.11. *A n^{-c_0}-approximate $\frac{c\log n}{\log(1/\varepsilon)}$-wise independent random string of size n on an alphabet of size $O(\varepsilon^{-6})$ satisfies ε-matching property with a non-zero constant probability. c and c_0 are sufficiently large constants.*

Proof. Let S be a pseudo-random string of length n with n^{-c_0}-approximate $\frac{c\log n}{\log(1/\varepsilon)}$-wise independent symbols. Then, Step (3.12) is invalid as the proposed upper-bound does not work for $l > \frac{c\log n}{\varepsilon\log(1/\varepsilon)}$. To bound the probability of intervals of size $\Omega\left(\frac{c\log n}{\varepsilon\log(1/\varepsilon)}\right)$ not satisfying ε-self matching property, we claim the following.

Lemma 3.5.12. *Any string of size $l > 100m$ which contains an ε-self-matching contains two sub-intervals I_1 and I_2 of size m where there is a matching of size $0.99\frac{m\varepsilon}{2}$ between I_1 and I_2.*

Using Lemma 3.5.12, one can conclude that any string of size $l > 100\frac{c\log n}{\varepsilon\log(1/\varepsilon)}$ which contains an ε-self-matching contains two sub-intervals I_1 and I_2 of size $\frac{c\log n}{\varepsilon\log(1/\varepsilon)}$ where there is a matching of size $\frac{c\log n}{2\log(1/\varepsilon)}$ between I_1 and I_2. Then, Step (3.12) can be revised by bounding above the probability of a long interval having an ε-self-matching by a union bound over the probability of pairs of its subintervals having a dense matching. Namely, for $l > 100\frac{c\log n}{\varepsilon\log(1/\varepsilon)}$, let us denote the event of $S[i, i+l)$ containing a ε-self-matching by $A_{i,l}$. Then, for $c_0 > 3c$,

$$\begin{aligned}
\Pr[A_{i,l}] &\leq \Pr\left[S \text{ contains } I_1,\, I_2 : |I_i| = \frac{c\log n}{\varepsilon\log(1/\varepsilon)} \text{ and } \mathrm{LCS}(I_1, I_2) \geq 0.99\frac{c\log n}{2\log(1/\varepsilon)}\right] \\
&\leq n^2 \binom{\varepsilon^{-1}c\log n/\log(1/\varepsilon)}{0.99c\log n/2\log(1/\varepsilon)}^2 \left[\left(\frac{1}{|\Sigma|}\right)^{\frac{0.99c\log n}{2\log(1/\varepsilon)}} + n^{-c_0}\right] \\
&\leq n^2 \left(2.04e\varepsilon^{-1}\right)^{\frac{2\times 0.99c\log n}{2\log(1/\varepsilon)}} 2\varepsilon^{\frac{6\times 0.99c\log n}{2\log(1/\varepsilon)}} \\
&= 2n^2 \left(2.04e\right)^{\frac{0.99c\log n}{\log(1/\varepsilon)}} \varepsilon^{\frac{4\times 0.99c\log n}{2\log(1/\varepsilon)}} \\
&= 2n^{2+\frac{c\ln(2.04e)}{\log(1/\varepsilon)}-1.98c} = O\left(n^{-c'}\right)
\end{aligned}$$

where first inequality follows from the fact there can be at most n^2 pairs of intervals of size $\frac{c\log n}{\varepsilon\log(1/\varepsilon)}$ in S and the number of all possible matchings of size $\frac{c\log n}{\log(1/\varepsilon)}$ between them is at most $\binom{\varepsilon^{-1}c\log n/\log(1/\varepsilon)}{c\log n/2\log(1/\varepsilon)}^2$. Further, for small enough ε, constant c' can be as large as one

desires by setting constant c large enough. Thus, Step (3.12) can be revised as:

$$\begin{aligned}\mathbb{E}[\gamma'] &\leq \sum_{l=\varepsilon^{-1}}^{\frac{100c\log n}{\varepsilon\log(1/\varepsilon)}} l\cdot\left[\left(\frac{e}{\varepsilon\sqrt{|\Sigma|}}\right)^{2\varepsilon}\right]^l + \sum_{l=\frac{100c\log n}{\varepsilon\log(1/\varepsilon)}}^{n} l\Pr[A_{i,l}] \\ &\leq \sum_{l=\varepsilon^{-1}}^{\infty} l\cdot\left[\left(\frac{e}{\varepsilon\sqrt{|\Sigma|}}\right)^{2\varepsilon}\right]^l + n^2\cdot O(n^{-c'}) \leq O(\varepsilon + n^{2-c'})\end{aligned}$$

For an appropriately chosen c, $2-c'<0$; hence, the second term vanishes as n grows. Therefore, the conclusion $\mathbb{E}[\gamma]\leq O(\varepsilon)$ holds for the $n^{-O(1)}$-approximate $\frac{\log n}{\log(1/\varepsilon)}$-wise independent string as well. □

Proof of Lemma 3.5.12. Let M be a monotone matching of size $l\varepsilon$ or more between S and itself containing only bad edges. We chop S into $\frac{l}{m}$ intervals of size m. On the one hand, the size of M is greater than $l\varepsilon$ and on the other hand, we know that the size of M is exactly $\sum_{i,j}|E_{i,j}|$ where $E_{i,j}$ denotes the number of edges between interval i and j. Thus:

$$l\varepsilon \leq \sum_{i,j}|E_{i,j}| \Rightarrow \frac{\varepsilon}{2} \leq \frac{\sum_{i,j}|E_{i,j}|/m}{2l/m}$$

Note that $\frac{|E_{i,j}|}{m}$ represents the density of edges between interval i and interval j. Further, Since M is monotone, there are at most $\frac{2l}{m}$ intervals for which $|E_{i,j}|\neq 0$ and subsequently $\frac{|E_{i,j}|}{m}\neq 0$. Hence, on the right hand side we have the average of $\frac{2l}{m}$ non-zero terms which is greater than $\varepsilon/2$. So, there have to be some i' and j' for which:

$$\frac{\varepsilon}{2} \leq \frac{|E_{i',j'}|}{m} \Rightarrow \frac{m\varepsilon}{2} \leq |E_{i',j'}|$$

To analyze more accurately, if l is not divisible by m, we simply throw out up to m last elements of the string. This may decrease ε by $\frac{m}{l}<\frac{\varepsilon}{100}$. □

Note that using the polynomial sample spaces of [NN93] Theorem 3.5.11 directly leads to a deterministic algorithm for finding a string of size n with ε-self-matching property. One simply has to check all possible points in the sample space of the $n^{-O(1)}$-approximate $\frac{c\log n}{\log(1/\varepsilon)}$-wise independent strings and finds a string S with $\gamma_S\leq\mathbb{E}[\gamma]=O(\varepsilon)$. In other words, using a brute-force complete search, one can find a string satisfying $O(\varepsilon)$-self-matching property in $O\left(|\Sigma|^{\frac{c\log n}{\log(1/\varepsilon)}}\right)=n^{O(1)}$ time.

Theorem 3.5.13. *There is a deterministic algorithm running in $n^{O(1)}$ time that finds a string of length n satisfying ε-self-matching property over an alphabet of size $O(\varepsilon^{-6})$.*

3.5.3 Global Repositioning Algorithm for Insdel Errors

Now, we provide an alternative repositioning algorithms to be used along with ε-self-matching strings (and therefore ε-synchronization strings) to form indexing solutions. Throughout the following sections, we let ε-synchronization string S be sent as the synchronization string in an instance of (n, δ)-indexing problem and string S' be received at the receiving end after going under up to $n\delta$ insertions or deletions.

The algorithm works as follows. On the first round, the algorithm finds the longest common subsequence between S and S'. Note that this common subsequence corresponds to a monotone matching M_1 between S and S'. On the next round, the algorithm finds the longest common subsequence between S and the subsequence of unmatched elements of S' (i.e., those that have not appeared in M_1). This common subsequence corresponds to a monotone matching between S and the elements of S' that do not appear in M_1. The algorithm repeats this procedure $\frac{1}{\beta}$ times to obtain $M_1, \cdots, M_{1/\beta}$ where β is a parameter that we will fix later. In the output of this repositioning algorithm, the position of $S'[j]$ is guessed as $S[i]$ if $S'[j]$ and $S[i]$ correspond to each other under one of the $1/\beta$ common subsequences that the algorithm finds. Otherwise, the algorithm declares '$\perp$'. (i.e., an "I don't know") A formal description of the algorithm can be found in Algorithm 3.

Note that the longest common subsequence of two strings of length $O(n)$ can be found in $O(n^2)$ time using dynamic programming. Therefore, the whole algorithm runs in $O\left(n^2/\beta\right)$. Now we proceed to analyzing the performance of the algorithm by bounding the number of misdecodings.

Algorithm 3 Global Repositioning Algorithm for Insertions and Deletions

Input: S, S'

1: **for** $i = 1$ to $|S'|$ **do**
2: $\quad Position[i] \leftarrow \perp$
3: **for** $i = 1$ to $\frac{1}{\beta}$ **do**
4: $\quad$ Compute $\mathrm{LCS}(S, S')$
5: $\quad$ **for all** Corresponding $S[i]$ and $S'[j]$ in $\mathrm{LCS}(S, S')$ **do**
6: $\quad\quad Position[j] \leftarrow i$
7: $\quad$ Remove all elements of $\mathrm{LCS}(S, S')$ from S'

Output: *Position*

Theorem 3.5.14. *Let d_i and d_r denote the number of symbols inserted into and deleted from the communication respectively. The global repositioning algorithm formalized in Algorithm 3 guarantees a maximum misdecoding count of $(n+d_i-d_r)\beta+\frac{\varepsilon}{\beta}n$. More specifically, for $\beta = \sqrt{\varepsilon}$, the number of misdecodings is no larger than $3n\sqrt{\varepsilon}$ and running time will be $O\left(n^2/\sqrt{\varepsilon}\right)$.*

Proof. First, we claim that at most $(n+d_i-d_r)\beta$ of the symbols that have been successfully transmitted are not matched in any of $M_1, \cdots, M_{1/\beta}$. Assume by contradiction that more

than $(n + d_i - d_r)\beta$ of the symbols that pass through the channel successfully are not matched in any of $M_1, \cdots, M_{1/\beta}$. Then, there exists a monotone matching of size greater than $(n + d_i - d_r)\beta$ between the unmatched elements of S' and S after $\frac{1}{\beta}$ rounds of finding and removing the longest common subsequence. This implies that each M_i has a size of $(n+d_i-d_r)\beta$ or more. Therefore, the summation of their sizes exceeds $(n+d_i-d_r)\beta \times \frac{1}{\beta} = n + d_i - d_r = |S'|$ which is impossible.

Furthermore, as a result of Theorem 3.5.4, each M_i contains at most εn incorrectly matched elements. Hence, no more than $\frac{\varepsilon}{\beta}n$ of the matched symbols are matched to an incorrect element (i.e., lead to and incorrect decoding). Therefore, the total number of misdecodings can be bounded by $(n + d_i - d_r)\beta + \frac{\varepsilon}{\beta}n$. □

3.5.4 Global Repositioning Algorithm for Deletion Errors

We now introduce a very simple linear time streaming repositioning algorithm that guarantees no more than $\frac{\varepsilon}{1-\varepsilon} \cdot n\delta$ misdecodings.

Let d_r denote the number of symbols removed by the adversary. As adversary is restricted to symbol deletions, each symbol received at the receiver corresponds to a symbol sent by the sender. Hence, there exists a monotone matching of size $|S'| = n' = n - d_r$ like $M = \{(t_1, 1), (t_2, 2), \cdots, (t_{n-d_r}, n - d_r)\}$ between S and S' which matches each of the received symbols to their actual positions.

Our simple streaming algorithm greedily matches S' to its left-most appearance in S as a subsequence. More precisely, the algorithm matches $S'[1]$ to $S[t'_1]$ where t'_1 is the smallest number where $S[t'_1] = S'[1]$. Then, the algorithm matches $S'[2]$ to the smallest $t'_2 > t'_1$ where $S[t'_2] = S'[2]$ and construct the whole matching M' by repeating this procedure. Note that as there is a matching of size $|S'|$ between S and S', the size of the resulting matching M' will be $|S'|$ as well. We claim that the following holds for this algorithm:

Theorem 3.5.15. *Any ε-synchronization string along with the algorithm described in Section 3.5.4 form a linear-time streaming solution for deletion-only (n, δ)-indexing problem guaranteeing $\frac{\varepsilon}{1-\varepsilon} \cdot n\delta$ misdecodings.*

Proof. This algorithm clearly works in a streaming manner and runs in linear time. To analyze the performance, we make use of the fact that M and M' are both monotone matchings of size $|S'|$ between S and S'. Therefore, $\bar{M} = \{(t_1, t'_1), (t_2, t'_2), \cdots, (t_{n-d_r}, t'_{n-d_r})\}$ is a monotone matching between S and itself. Note that if $t_i \neq t'_i$, then the algorithm has incorrectly decoded the index symbol i. Let p be the number of all such symbols. Then matching $\bar{M}$ consists of $n - d_r - p$ good pairs and p bad pairs. Therefore, using Theorem 3.5.2 we have the following.

$$p \leq \varepsilon(n - (n - d_r - p)) \Rightarrow p \leq \varepsilon(d_r + p) \Rightarrow p < \frac{\varepsilon}{1 - \varepsilon} \cdot d_r$$

□

3.5.5 Global Repositioning Algorithm for Insertion Errors

We now consider another simplified case where adversary is restricted to only inserting symbols. We propose a decoding algorithm whose output is guaranteed to be error-free and to contain less than $\frac{n\delta}{1-\varepsilon}$ misdecodings.

Assume that d_i symbols are inserted into the string S to turn it in into S' of size $n+d_i$ on the receiving side. Again, it is clear that there exists a monotone matching M of size n like $M = \{(1,t_1),(2,t_2),\cdots,(n,t_n)\}$ between S and S' that matches each symbol in S' to its actual position in S.

The repositioning algorithm we present, matches $S[i]$ to $S'[t'_i]$ in its output, M', if and only if in all possible monotone matchings between S and S' that saturate S (i.e., are of size $|S| = n$), $S[i]$ is matched to $S'[t'_i]$. Note that any symbol $S[i]$ that is matched to $S'[t'_i]$ in M' has to be matched to the same element in M; therefore, the output of this algorithm does not contain any incorrectly decoded indices; therefore, the algorithm is error-free.

Now, we are going to first provide a linear time approach to implement this algorithm and then prove an upper-bound of $\frac{d_i}{1-\varepsilon}$ on the number of misdecodings. To this end, we make use of the following lemma:

Lemma 3.5.16. *Let $M_L = \{(1,l_1),(2,l_2),\cdots,(n,l_n)\}$ be the monotone matching between S and S' such that yields the smallest lexicographical value for $l_1,\cdots,l_n$. We call M_L the* left-most *matching between S and S'. Similarly, let $M_R = \{(1,r_1),\cdots,(n,r_n)\}$ be the monotone matching for which $r_n,\cdots,r_1$ yields the largest possible lexicographical value. Then $S[i]$ is matched to $S'[t'_i]$ in all possible monotone matchings of size n between S and S' if and only if $(i,t'_i) \in M_R \cap M_L$.*

This lemma can be proved by a simple contradiction argument. Our algorithm starts by computing left-most and right-most monotone matchings between S and S' using the straightforward greedy algorithm introduced in Section 3.5.4 on (S,S') and their reversed versions. It then outputs the intersection of these two matchings as the answer.

This algorithm runs in linear time since the task of finding the left-most and right-most matchings are done in linear time. To analyze this algorithm, we bound above the number of successfully transmitted symbols that the algorithm refuses to decode, denoted by p. To do so, we make use of the fact that $n-p$ elements of S' are matched to the same element of S in both M_L and M_R. As there are p elements in S that are matched to different elements in S' and there is a total of $n+d_i$ elements in S', there has to be at least $2p-[(n+d_i)-(n-p)] = p-d_i$ elements in S' that are matched to different elements of S under M_L and M_R.

Consider the following monotone matching from S to itself.

$$\begin{aligned} M \;=\; & \{(i,i) : \text{If } S[i] \text{ is matched to the same position of } S' \text{ in both } M \text{ and } M'\} \\ & \cup \{(i,j) : \exists k \text{ s.t. } (i,k) \in M_L, (j,k) \in M_R\} \end{aligned}$$

Note that monotonicity follows the fact that both M_L and M_R are both monotone matchings between S and S'. We have shown that the size of the second set is at least $p-d_i$ and the size of the first set is by definition $n-p$. Also, all pairs in the first set are good

pairs and all in the second one are bad pairs. Therefore, Theorem 3.5.2 implies that

$$(p - d_i) \leq \varepsilon(n - (n - p)) \Rightarrow p < \frac{d_i}{1 - \varepsilon}$$

which proves the efficiency claim and gives the following theorem.

Theorem 3.5.17. *Any ε-synchronization string along with the algorithm described in Section 3.5.5 form a linear-time error-free solution for insertion-only (n, δ)-indexing problem guaranteeing $\frac{1}{1-\varepsilon} \cdot n\delta$ misdecodings.*

Finally, we remark that a similar non-streaming algorithm can be applied to the case of deletion-only errors. Namely, one can compute the left-most and right-most matchings between the received string and string that is supposed to be received and output the common edges. By a similar argument as above, one can prove the following:

Theorem 3.5.18. *Any ε-synchronization string along with the algorithm described in Section 3.5.5 form a linear-time error-free solution for deletion-only (n, δ)-indexing problem guaranteeing $\frac{\varepsilon}{1-\varepsilon} \cdot n\delta$ misdecodings.*

3.5.6 Linear-Time Near-MDS Insertion-Only and Deletion-Only Codes

In the same manner as in Theorem 3.1.1, we can use error-free indexing solutions presented in Theorems 3.5.17 and 3.5.18 along with near-MDS (erasure) correcting codes to derive the following linear-time insertion-only or deletion-only errors.

Theorem 3.5.19. *For any $\varepsilon > 0$ and $\delta \in (0, 1)$:*

- *There exists an encoding map $E : \Sigma^k \to \Sigma^n$ and a decoding map $D : \Sigma^* \to \Sigma^k$ such that if x is a subsequence of $E(m)$ where $|x| \geq n - n\delta$ then $D(x) = m$. Further $\frac{k}{n} > 1 - \delta - \varepsilon$, $|\Sigma| = f(\varepsilon)$, and E and D are explicit and have linear running times in n.*
- *There exists an encoding map $E : \Sigma^k \to \Sigma^n$ and a decoding map $D : \Sigma^* \to \Sigma^k$ such that if $E(m)$ is a subsequence of x where $|x| \leq n + n\delta$ then $D(x) = m$. Further $\frac{k}{n} > 1 - \delta - \varepsilon$, $|\Sigma| = f(\varepsilon)$, and E and D are explicit and have linear running times in n.*

3.5.7 Repositioning Using the Relative Suffix Pseudo-Distance (RSPD)

In this section, we show how one can slightly improve the constants in the results obtained in Section 3.4.2 by replacing RSD with a related notion of "distance" between two strings introduced in [BGMO17]. We call this notion the *relative suffix pseudo-distance* or RSPD both to distinguish it from our RSD relative suffix distance and also because RSPD is not a metric–it is neither symmetric nor satisfies the triangle inequality.

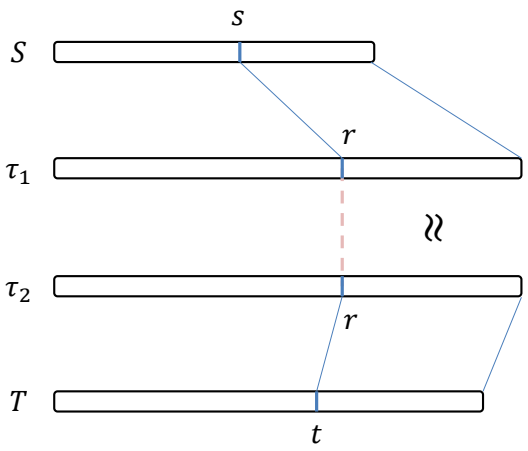


Figure 3.2: Pictorial representation of the notation used in Lemma 3.5.22

Definition 3.5.20 (Relative Suffix Pseudo-Distance (RSPD))**.** *Given any two strings* $c, \tilde{c} \in \Sigma^*$*, the* relative suffix pseudo-distance *between* c *and* $\tilde{c}$ *is*

$$\mathrm{RSPD}\left(c, \tilde{c}\right) = \min_{\tau : c \to \tilde{c}} \left\{ \max_{i=1}^{|\tau_1|} \left\{ \frac{sc\left(\tau_1\left[i, |\tau_1|\right]\right) + sc\left(\tau_2\left[i, |\tau_2|\right]\right)}{|\tau_1| - i + 1 - sc\left(\tau_1\left[i, |\tau_1|\right]\right)} \right\} \right\}$$

We derive our repositioning algorithm by proving the following useful property of ε-synchronization strings.

Lemma 3.5.21. *Let* $S \in \Sigma^n$ *be an* ε*-synchronization string and* $\tilde{c} \in \Sigma^m$*. Then there exists at most one* $c \in \bigcup_{i=1}^{n} S[1..i]$ *such that* $\mathrm{RSPD}(c, \tilde{c}) \leq 1 - \varepsilon$*.*

Before proceeding to the proof of Lemma 3.5.21, we prove the following lemma.

Lemma 3.5.22. *Let* $\mathrm{RSPD}(S, T) \leq 1 - \varepsilon$*, then:*

1. *For every* $1 \leq s \leq |S|$*, there exists* t *such that*

$$\mathrm{ED}\left(S[s, |S|], T\left[t, |T|\right]\right) \leq (1 - \varepsilon)(|S| - s + 1).$$

2. *For every* $1 \leq t \leq |T|$*, there exists* s *such that*

$$\mathrm{ED}\left(S[s, |S|], T\left[t, |T|\right]\right) \leq (1 - \varepsilon)(|S| - s + 1).$$

Proof. Each part will be proved separately.

Part 1. Let τ be the string matching that attains the minimization in the definition of $\mathrm{RSPD}(S, T)$. There exist some r such that $\mathrm{del}(\tau_1\left[r, |\tau_1|\right]) = S[s, |S|]$. Note

that $\text{del}(\tau_2[r,|\tau_2|])$ is a suffix of T. Therefore, there exists some t such that $T[t,|T|] = \text{del}(\tau_2[r,|\tau_2|])$. Now,

$$\begin{aligned}
\text{ED}(S[s,|S|],T[t,|T|]) &\leq sc(\text{del}(\tau_1[r,|\tau_1|])) + sc(\text{del}(\tau_2[r,|\tau_1|])) \\
&= \frac{sc(\text{del}(\tau_1[r,|\tau_1|])) + sc(\text{del}(\tau_2[r,|\tau_1|]))}{|\tau_1| - r + 1 - sc(\tau_1[r,|\tau_1|])} \\
&\quad \cdot(|\tau_1| - r + 1 - sc(\tau_1[r,|\tau_1|])) \qquad (3.13) \\
&\leq \text{RSPD}(S,T)\cdot(|S| - s + 1) \\
&\leq (1-\varepsilon)\cdot(|S| - s + 1) \qquad (3.14)
\end{aligned}$$

Part 2. Similarly, let τ be the string matching yielding $\text{RSPD}(S,T)$. There exists some r such that $\text{del}(\tau_2[r,|\tau_2|]) = T[t,|T|]$. Now, $\text{del}(\tau_1[r,|\tau_1|])$ is a suffix of S. Therefore, there exists some s such that $S[s,|S|] = \text{del}(\tau_1[r,|\tau_1|])$. Now, all the steps we took to prove equation (3.14) hold and the proof is complete. □

Algorithm 4 Minimum RSPD Decoding Algorithm (Guessing the position of the last symbol of the received string)

Input: A received message $\tilde{c} \in \Sigma^m$ and an ε-synchronization string $S \in \Sigma^n$

1: $ans \leftarrow \emptyset$
2: **for** Any prefix c of S **do**
3: $\quad d[i][j][l] \leftarrow \min_{\substack{\tau: c(i)\to\tilde{c}(j) \\ sc(\tau_1)=l}} \max_{k=1}^{|\tau_1|} \frac{sc(\tau_1[k..|\tau_1|])+sc(\tau_2[k..|\tau_2|])}{|\tau_1|-k+1+sc(\tau_1[k..|\tau_1|])}$
4: $\quad \text{RSPD}(c,\tilde{c}) \leftarrow \min_{l'=0}^{|\tilde{c}|} d[\mathtt{i}][|\tilde{c}|][l']$
5: $\quad$ **if** $\text{RSPD}(c,\tilde{c}) \leq 1-\varepsilon$ **then**
6: $\quad\quad ans \leftarrow c$

Output: ans

Proof of Lemma 3.5.21. For a contradiction, suppose that there exist $\tilde{c}$, l and l' such that $l < l'$, $\text{RSPD}(S[1,l],\tilde{c}) \leq 1-\varepsilon$ and $\text{RSPD}(S[1,l'],\tilde{c}) \leq 1-\varepsilon$. Now, using part 1 of Lemma 3.5.22, there exists k such that $\text{ED}\left(S[l+1,l'],\tilde{c}[k,|\tilde{c}|]\right) \leq (1-\varepsilon)(l'-l)$. Further, part 2 of Lemma 3.5.22 gives that there exist l'' such that $\text{ED}\left(S[l''+1,l],\tilde{c}[k,|\tilde{c}|]\right) \leq (1-\varepsilon)(l-l'')$. Hence,

$$\text{ED}(S[l{+}1,l'],S[l'{+}1,l'']) \leq \text{ED}(S[l{+}1,l'],\tilde{c}[k,|\tilde{c}|]){+}\text{ED}(S[l'{+}1,l''],\tilde{c}[k,|\tilde{c}|]) \leq (1{-}\varepsilon)(l'{-}l'')$$

which contradicts the fact that S is an ε-synchronization string. □

In the same spirit as Section 3.4.2, one can develop a repositioning algorithm based on Lemma 3.5.21 that works as follows: upon arrival of each symbol, find a prefix of synchronization index string S that has the smallest RSPD to the string that is received

so far (denoted by $\tilde{c}$). We call this algorithm the minimum RSPD decoding algorithm. Theorems 3.5.23 and 3.5.24 describe the computational complexity and misdecoding count of such repositioning algorithm.

Theorem 3.5.23. *Let $S \in \Sigma^n$ be an ε-synchronization string, and $\tilde{c} \in \Sigma^m$. Then Algorithm 4, given input S and $\tilde{c}$, either returns the unique prefix c of S such that $\mathrm{RSPD}(c, \tilde{c}) \leq 1 - \varepsilon$ or returns $\perp$ if no such prefix exists. Moreover, Algorithm 4 runs in $O(n^4)$ time. Therefore, using it over each received symbol to guess the position of all symbols of a communication, one derives a repositioning algorithm that runs in $O(n^5)$ time.*

Proof. To find c, we calculate the RSPD of $\tilde{c}$ and all prefixes of S one by one. We only need to show that the RSPD of two strings of length at most n can be found in $O(n^3)$. We do this using dynamic programming. Let us try to find $\mathrm{RSPD}(s, t)$. Further, let $s(i)$ represent the suffix of s of length i and $t(j)$ represent the suffix of t of length j. Now, let $d[i][j][l]$ be the minimum string matching (τ_1, τ_2) from $s(i)$ to $t(j)$ such that $sc(\tau_1) = l$. In other words,

$$d[i][j][l] = \min_{\substack{\tau : s(i) \to t(j) \\ sc(\tau_1) = l}} \max_{k=1}^{|\tau_1|} \frac{sc\left(\tau_1\left[k..|\tau_1|\right]\right) + sc\left(\tau_2\left[k..|\tau_2|\right]\right)}{|\tau_1| - k + 1 + sc\left(\tau_1\left[k..|\tau_1|\right]\right)},$$

where τ is a string matching for $s(i)$ and $t(j)$. Note that for any $\tau : s(i) \to t(j)$, one of the following three scenarios might happen:

1. $\tau_1(1) = \tau_2(1) = s\left(|s| - (i-1)\right) = t(|t| - (j-1))$: In this case, removing the first elements of τ_1 and τ_2 gives a valid string matching from $s(i-1)$ to $t(j-1)$.

2. $\tau_1(1) = *$ and $\tau_2(1) = t(|t| - (j-1))$: In this case, removing the first element of τ_1 and τ_2 gives a valid string matching from $s(i)$ to $t(j-1)$.

3. $\tau_2(1) = *$ and $\tau_1(1) = s(|s| - (i-1))$: In this case, removing the first element of τ_1 and τ_2 gives a valid string matching from $s(i-1)$ to $t(j)$.

This implies that

$$d[i][j][l] = \min \Bigg\{ \begin{array}{l} d[i-1][j-1][l] \text{ (Only if } s(i) = t(j)), \\ \max\left\{ d[i][j-1][l-1], \dfrac{l + (j - (i-l))}{(i+l) + l} \right\}, \\ \max\left\{ d[i-1][j][l], \dfrac{l + (j - (i-l))}{(i+l) + l} \right\} \Bigg\}. \end{array}$$

Hence, one can find $\mathrm{RSPD}(s, t)$ by minimizing $d[|s|][|t|][l]$ over all possible values of l, as Algorithm 4 does in Step 4 for all prefixes of S. Finally, Algorithm 4 returns the prefix c such that $\mathrm{RSPD}(c, \tilde{c}) \leq 1 - \varepsilon$ if one exists, and otherwise it returns $\perp$. □

We conclude by showing that if an ε-synchronization string of length n is used along with the minimum RSPD algorithm, the number of misdecodings will be at most $\frac{n\delta}{1-\varepsilon}$.

Theorem 3.5.24. *Suppose that S is an ε-synchronization string of length n over alphabet Σ that is sent over an insertion-deletion channel with c_i insertions and c_d deletions. Repositioning via using Algorithm 4 to guess the position of each received symbol results into no more than $\frac{c_i}{1-\varepsilon} + \frac{c_d\varepsilon}{1-\varepsilon}$ misdecodings.*

Proof. The proof of this theorem is similar to the proof of Theorem 3.4.10. Let prefix $S[1,i]$ be sent through the channel and $S_\tau[1,j]$ be received on the other end as the result of adversary's set of actions τ. Further, assume that $S_\tau[j]$ is successfully transmitted and is actually $S[i]$ sent by the other end. We first show that $\mathrm{RSPD}(S[1,i], S'[1,j])$ is less than the relative suffix error density:

$$\begin{aligned}
\mathrm{RSPD}(S[1,i], S'[1,j]) &= \min_{\tilde{\tau}:c\to\tilde{c}} \left\{ \max_{k=1}^{|\tilde{\tau}_1|} \left\{ \frac{sc\left(\tilde{\tau}_1\left[k,|\tilde{\tau}_1|\right]\right) + sc\left(\tilde{\tau}_2\left[k,|\tilde{\tau}_2|\right]\right)}{|\tilde{\tau}_1| - k + 1 - sc\left(\tilde{\tau}_1\left[k,|\tilde{\tau}_1|\right]\right)} \right\} \right\} \\
&\leq \max_{k=1}^{|\tau_1|} \left\{ \frac{sc\left(\tau_1\left[k,|\tau_1|\right]\right) + sc\left(\tau_2\left[k,|\tau_2|\right]\right)}{|\tau_1| - k + 1 - sc\left(\tau_1\left[k,|\tau_1|\right]\right)} \right\} \\
&= \max_{j\leq i} \frac{\mathcal{E}(j,i)}{i-j} = \text{ Relative Suffix Error Density}
\end{aligned}$$

Now, using Theorem 3.4.14, we know that the relative suffix error density is smaller than $1-\varepsilon$ upon arrival of all but at most $\frac{c_i+d_d}{1-\varepsilon} - c_d$ of successfully transmitted symbols. Along with Lemma 3.5.21, this results into the conclusion that the minimum RSPD decoding algorithm guarantees no more than $\frac{c_i}{1-\varepsilon} + c_d\left(\frac{1}{1-\varepsilon} - 1\right)$ misdecodings. □

Chapter 4

List-Decodable Codes via Indexing

In this chapter, we study codes that are list-decodable under insertions and deletions. Specifically, we consider the setting where, given a codeword x of length n over some finite alphabet Σ of size q, $\delta \cdot n$ codeword symbols may be adversarially deleted and $\gamma \cdot n$ symbols may be adversarially inserted to yield a corrupted word w. A code is said to be list-decodable if there is an (efficient) algorithm that, given w, reports a small list of codewords that include the original codeword x. Given δ and γ we study what is the rate R for which there exists a constant q and list size L such that there exist codes of rate R correcting δ-fraction insertions and γ-fraction deletions while reporting lists of size at most L.

Using the concept of synchronization strings, introduced in Chapter 3, we show some surprising results. We show that for every $0 \leq \delta < 1$, every $0 \leq \gamma < \infty$ and every $\varepsilon > 0$ there exist codes of rate $1 - \delta - \varepsilon$ and constant alphabet (so $q = O_{\delta,\gamma,\varepsilon}(1)$) and sub-logarithmic list sizes. Furthermore, these codes are accompanied by efficient (polynomial time) decoding algorithms. We stress that the fraction of insertions can be arbitrarily large (more than 100%), and the rate is independent of this parameter.

Our result sheds light on the remarkable asymmetry between the impact of insertions and deletions from the point of view of error-correction: Whereas deletions cost in the rate of the code, insertion costs are borne by the adversary and not the code! Our results also highlight the dominance of the model of insertions and deletions over the Hamming model: A Hamming error is equal to one insertion and one deletion (at the same location). Thus the effect of δ-fraction Hamming errors can be simulated by δ-fraction of deletions and δ-fraction of insertions — but insdel codes can deal with much more insertions without loss in rate (though at the price of higher alphabet size).

4.1 Introduction

We study the complexity of "insdel coding", i.e., codes designed to recover from insertion and deletion of characters, under the model of "list-decoding", i.e., when the decoding algorithm is allowed to report a (short) list of potential codewords that is guaranteed to include the transmitted word if the number of errors is small enough. The results presented in Chapter 3 have shown major progress leading to tight, or nearly tight, bounds on central parameters of codes (with efficient encoding and decoding algorithms as well) under the setting of *unique* decoding. This chapter complements the results of Chapter 3 by exploring the list-decoding versions of these questions. In the process, the results of this chapter also reveal some striking features of the insdel coding problem that were not exposed by previous works. To explain some of this, we introduce our model and lay out some of the context below.

4.1.1 Insdel Coding and List Decoding

The principal question we ask is "what is the *rate* of a code that can recover from γ fraction insertions and δ fraction deletions over a sufficiently large alphabet?". Once the answer to this question is determined we ask how small an alphabet suffices to achieve this rate. The terms "rate", "alphabet", and "recovery" are defined similar to the unique-decoding case. An insdel *encoder* over alphabet Σ of block length n is an injective function $E : \Sigma^k \to \Sigma^n$. The associated "code" is the image of the function C. The rate of a code is the ratio k/n. We say that an insdel code C is $(\gamma, \delta, L(n))$-list-decodable if there exists a function $D : \Sigma^* \to 2^C$ such that $|D(w)| \leq L(n)$ for every $w \in \Sigma^*$ and for every codeword $x \in C$ and every word w obtained from x be $\delta \cdot n$ deletions of characters in x followed by $\gamma \cdot n$ insertions, it is the case that $x \in D(w)$. In other words the list-decoder D outputs a list of at most $L(n)$ codewords that is guaranteed to include the transmitted word x if the received word w is obtained from x by at most δ-fraction deletions and γ-fraction insertions. Our primary quest in this chapter is the largest rate R for which there exists an alphabet of size $q \triangleq |\Sigma|$ and an infinite family of insdel codes of rate at least R, that are $(\gamma, \delta, L(n))$-list-decodable. Of course we are interested in results where $L(n)$ is very slowly growing with n (if at all). In the results below we get $L(n)$ which is polynomially large in terms of n. Furthermore, when a given rate is achievable we seek codes with efficient encoder and decoder (i.e., the functions E and D are polynomial time computable).

Previous Work. The list-decoding model was first introduced by Elias [Eli57] and Wozencraft [Woz58] for Hamming-type errors. Several breakthroughs for list-decoding from Hamming-type errors have been made ever since. Notably, the first efficient list-decoding procedure was designed for Reed-Solomon codes by Sudan [Sud97]. However, very few works have considered list-decoding for insertion-deletion codes. A work by Wachter-Zeh [WZ18] provides Johnson-like upper-bounds for insertions and deletions, i.e., bounds on the list size in terms of the minimum edit-distance of a given code. Moreover, for Varshamov-Tenengolts codes, [WZ18] presents lower bounds on the maximum list size as well as a list-decoding algorithm against a constant number of insertions and deletions.

Follow-up work by Hayashi and Yasunaga [HY18] corrected some subtle but crucial bugs in [WZ18] and reproved a corrected Johnson Bound for insdel codes. They also showed that the codes of [BGH17] could be list-decoded from a fraction ≈ 0.707 of insertions. Lastly, via a concatenation scheme used in [GW17, GL16] they furthermore made these codes efficient. A recent work of Liu, Tjuawinata, and Xing [LTX19] also provides efficiently list-decodable insertion-deletion codes and derives a Zyablov-type bound.

4.1.2 Our Results

We now present our results on the rate and alphabet size of insdel coding under list-decoding. Two points of contrast that we use below are corresponding bounds in (1) the Hamming error setting for list-decoding and (2) the insdel coding setting with unique-decoding.

Rate Under List Decoding

Our main theorem for list-decoding shows that, given $\gamma, \delta, \varepsilon \geq 0$ there is a $q = q_{\varepsilon,\gamma}$ and a slowly growing function $L = L_{\varepsilon,\gamma}(n)$ such that there are q-ary insdel codes that achieve a rate of $1 - \delta - \varepsilon$ that are $(\gamma, \delta, L(n))$-list decodable. Furthermore, the encoding and decoding are efficient! The formal statement of the main result is as follows.

Theorem 4.1.1. *For every $0 < \delta, \varepsilon < 1$ and $\gamma > 0$, there exist a family of list-decodable insdel codes that can protect against δ-fraction of deletions and γ-fraction of insertions and achieves a rate of at least $1-\delta-\varepsilon$ or more over an alphabet of size $\left(\frac{\gamma+1}{\varepsilon^2}\right)^{O\left(\frac{\gamma+1}{\varepsilon^3}\right)} = O_{\gamma,\varepsilon}(1)$. These codes are list-decodable with lists of size $L_{\varepsilon,\gamma}(n) = \exp\left(\exp\left(\exp\left(\log^* n\right)\right)\right)$, and have polynomial time encoding and decoding complexities.*

The rate in the theorem above is immediately seen to be optimal even for $\gamma = 0$. In particular an adversary that deletes the last $\delta \cdot n$ symbols already guarantees an upper bound on the rate of $1 - \delta$.

We now contrast the theorem above with the two contrasting settings listed earlier. Under unique decoding the best possible rate that can be achieved with δ-fraction deletions and γ-fraction insertions is upper bounded by $1 - (\gamma + \delta)$. Matching constructions have been achieved, only recently, by this body of work (see Chapter 3). In contrast our rate has no dependence on γ and thus dominates the above result. The only dependence on γ is in the alphabet size and list-size and we discuss the need for this dependence later below.

We now turn to the standard "Hamming error" setting: Here an adversary may change an arbitrary δ-fraction of the codeword symbols. In this setting it is well-known that given any $\varepsilon > 0$, there are constants $q = q(\varepsilon)$ and $L = L(\varepsilon)$ and an infinite family of q-ary codes of rate at least $1 - \delta - \varepsilon$ that are list-decodable from δ fraction errors with list size at most L. In a breakthrough from the last decade, Guruswami and Rudra [GR08] showed explicit codes that achieve this with efficient algorithms. The state-of-the-art results in this field yield list size $L(n) = o(\log^{(r)} n)$ for any integer r where $\log^{(r)}$ is the rth iterated logarithm and alphabet size $2^{\tilde{O}(\varepsilon^{-2})}$ [GX17], which are nearly optimal.

The Hamming setting with δ-fraction errors is clearly a weaker setting than the setting with δ-fraction deletions and $\gamma \geq \delta$ fraction of insertions in that an adversary of the latter kind can simulate the former. (A Hamming error is a deletion followed by an insertion at the same location.) The insdel setting is thus stronger in two senses: it allows $\gamma > \delta$ and gives greater flexibility to the adversary in choosing locations of insertions and deletions. Yet our theorem shows that the stronger adversary can still be dealt with, without qualitative changes in the rate. The only difference is in the dependence of q and L on γ, which we discuss next.

We briefly remark at this stage that, while this chapter simultaneously "dominates" the results of Chapter 3 of this body of work as well as Guruswami and Rudra [GR08], this happens because we use both of these as ingredients in this chapter. We elaborate further on this in Section 4.3. Indeed our first result (see Theorem 4.3.1) shows how we can obtain Theorem 4.1.1 by using capacity achieving "list-recoverable codes" in combination with synchronization strings in a modular fashion.

We will show in Chapter 9 that codes from Theorem 4.1.1 can be used to reduce the decoding complexity of the Singleton-bound achieving codes from Theorem 3.1.1 to near-linear time.

4.2 Definitions and Preliminaries

4.2.1 Synchronization Strings

In this section, we briefly recapitulate synchronization strings and core ideas around coding via indexing from Chapter 3 for the sake of convenience. Readers familiar with Chapter 3 may skip this section.

Introduced in Chapter 3, synchronization strings are mathematical objects that are very useful in overcoming synchronization errors. The general idea to obtain resilience against synchronization errors in various communication setups is to index each symbol of the communication with symbols of a synchronization string and then *guessing* the actual position of received symbols on the other side using the indices.

Suppose that two parties are communicating over a channel that suffers from α-fraction of insertions and deletions and one of the parties sends a pre-shared string S of length n to the other one. A distorted version of S will arrive at the receiving end that we denote by S'. A symbol $S[i]$ is called to be a *successfully transmitted* symbol if it is not removed by the adversary. A *repositioning algorithm* on the receiving side is an algorithm that, for any received symbol, guesses its actual position in S by either returning a number in $[1..n]$ or $\top$ which means the algorithm is not able to guess the position of that symbol. For such a decoding algorithm, a successfully transmitted symbol whose index is not guessed correctly by the decoding algorithm is called a *misdecoding*. For the sake of convenience, we restate some of the useful definitions and theorems from Chapter 3.

Definition 4.2.1 (ε-synchronization string (Definition 3.4.4))**.** *String $S \in \Sigma^n$ is an ε-synchronization string if for every $1 \leq i < j < k \leq n+1$ we have that* $\mathrm{ED}\left(S[i,j), S[j,k)\right) > (1-\varepsilon)(k-i)$.

It is shown in Chapter 3 that ε-synchronization strings exist over alphabets of size $\mathsf{poly}(\varepsilon^{-1})$ and can be efficiently constructed. An important property of ε-synchronization strings discussed in Chapter 3 is the self matching property defined as follows.

Definition 4.2.2 (ε-self-matching property (Definition 3.5.3))**.** *String S satisfies ε-self-matching property if for any two sequences of indices $1 \leq a_1 < a_2 < \cdots < a_k \leq |S|$ and $1 \leq b_1 < b_2 < \cdots < b_k \leq |S|$ that satisfy $S[a_i] = S[b_i]$ and $a_i \neq b_i$, k is smaller than $\varepsilon|S|$.*

In the end, we review the following theorem from Chapter 3 that shows the close connection between synchronization string property and the self-matching property.

Theorem 4.2.3 (Restatement of Theorem 3.5.4)**.** *If S is an ε-synchronization string, then all substrings of S satisfy ε-self-matching property.*

4.2.2 List Recoverable Codes

A code $\mathcal{C}$ given by the encoding function $\mathcal{E} : \Sigma^{nr} \to \Sigma^n$ is called to be (α, l, L)-list recoverable if for any collection of n sets $S_1, S_2, \cdots, S_n \subset \Sigma$ of size l or less, there are at most L codewords for which more than αn elements appear in the list that corresponds to their position, i.e.,

$$|\{x \in \mathcal{C} \mid |\{i \in [n] \mid x_i \in S_i\}| \geq \alpha n\}| \leq L.$$

The study of list-recoverable codes was inspired by Guruswami and Sudan's list-decoder for Reed-Solomon codes [GS99]. Since then, list-recoverable codes have became a very useful tool in coding theory [GI01, GI02, GI03, GI04] and there have been a variety of constructions provided for them by several works [GR08, GW11, GX13, Kop15, HW18, GX17, HRZW19, KRR+19]. In this chapter, we will make use of the following capacity-approaching polynomial-time list-recoverable codes given by Hemenway, Ron-Zewi, and Wootters [HRZW19] that is obtained by modifying the approach of Guruswami and Xing [GX13].

Theorem 4.2.4 (Hemenway et. al. [HRZW19, Theorem A.7])**.** *Let q be an even power of a prime, and choose $l, \epsilon > 0$, so that $q \geq \epsilon^{-2}$. Choose $\rho \in (0, 1)$. There is an $m_{min} = O(l \log_q(l/\epsilon)/\epsilon^2)$ so that the following holds for all $m \geq m_{min}$. For infinitely many n (all n of the form $q^{e/2}(\sqrt{q} - 1)$ for any integer e), there is a deterministic polynomial-time construction of an F_q-linear code $C : \mathbb{F}_{q^m}^{\rho n} \to \mathbb{F}_{q^m}^n$ of rate ρ and relative distance $1 - \rho - O(\epsilon)$ that is $(1 - \rho - \epsilon, l, L)$-list-recoverable in time $\mathsf{poly}(n, L)$, returning a list that is contained in a subspace over $\mathbb{F}_q$ of dimension at most $\left(\frac{l}{\epsilon}\right)^{2^{\log^*(mn)}}$.*

4.3 List Decoding for Insertions and Deletions

In this section, we prove Theorem 4.1.1 by constructing a list-decodable code of rate $1 - \delta - \varepsilon$ that provides resilience against $0 < \delta < 1$ fraction of deletions and γ fraction of insertions over a constant-sized alphabet. Our construction heavily relies on the following theorem that, in the same fashion as Chapter 3, uses the technique of indexing an error

correcting code with a synchronization string to convert a given list-recoverable code into an insertion-deletion code.

Theorem 4.3.1. *Let $\mathcal{C} : \Sigma^{nR} \to \Sigma^n$ be a (α, l, L)-list recoverable code with rate R, encoding complexity T_{Enc} and decoding complexity complexity T_{Dec}. For any $\varepsilon > 0$ and $\gamma \leq \frac{l\varepsilon}{2} - 1$, by indexing $\mathcal{C}$ with an $\frac{\varepsilon^2}{4(1+\gamma)}$-synchronization string, one can obtain an L-list decodable insertion-deletion code $\mathcal{C}' : \Sigma^{nr} \to [\Sigma \times \Gamma]^n$ that corrects from $\delta < 1 - \alpha - \varepsilon$ fraction of deletions and γ fraction of insertions where $|\Gamma| = (\varepsilon^2/(1+\gamma))^{-O(1)}$. $\mathcal{C}'$ is encodable and decodable in $O(T_{Enc} + n)$ and $O(T_{Dec} + n^2(1+\gamma^2)/\varepsilon)$ time respectively.*

We take two major steps to prove Theorem 4.3.1. In the first step (Theorem 4.3.3), we use the synchronization string indexing technique from Chapter 3 and show that by indexing the symbols that are conveyed through an insertion-deletion channel with symbols of a synchronization string, the receiver can make *lists* of candidates for any position of the sent string such that $1 - \delta - \varepsilon$ fraction of lists are guaranteed to contain the actual symbol sent in the corresponding step and the length of the lists is guaranteed to be smaller than some constant $O_{\gamma,\varepsilon}(1)$.

In the second step, we use list-recoverable codes on top of the indexing scheme to obtain a list decoding using lists of candidates for each position produced by the former step.

We start by the following lemma that directly implies the first step stated in Theorem 4.3.3.

Lemma 4.3.2. *Assume that a sequence of n symbols denoted by $x_1 x_2 \cdots x_n$ is indexed with an ε-synchronization string and is communicated through a channel that suffers from up to δn deletions for some $0 \leq \delta < 1$ and γn insertions. Then, on the receiving end, it is possible to obtain n lists $A_1, \cdots, A_n$ such that, for any desired integer K, for at least $n \cdot \left(1 - \delta - \frac{1+\gamma}{K} - K \cdot \varepsilon\right)$ of them, $x_i \in A_i$. All lists contain up to K elements and the average list size is at most $1 + \gamma$. These lists can be computed in $O\left(K(1+\gamma)n^2\right)$ time.*

Proof. The decoding algorithm we propose to obtain the lists that satisfy the guarantee promised in the statement is the global algorithm introduced in Theorem 3.5.14.

Let S be the ε-synchronization string used for indexing and S' be the index portion of the received string on the other end. Note that S is pre-shared between the sender and the receiver. The decoding algorithm starts by finding a longest common substring M_1 between S and S' and adding the position of any matched element from S' to the list that corresponds to its respective match from side S. Then, it removes every symbol that have been matched from S' and repeats the previous step by finding another longest common subsequence M_2 between S and the remaining elements of S'. This procedure is repeated K times to obtain $M_1, \cdots, M_K$. This way, lists A_i are formed by including every element in S' that is matched to $S[i]$ in any of $M_1, \cdots, M_K$.

A_i contains the actual element that corresponds to $S[i]$, denoted by $S'[j]$, if and only if $S[i]$ is successfully transmitted (i.e., not removed by the adversary), appears in one of M_ks, and matches to $S[i]$ in M_k. Hence, there are three scenarios under which A_i does not contain its corresponding element $S[i]$.

1. $S[i]$ gets deleted by the adversary.

2. $S[i]$ is successfully transmitted but, as $S'[j]$ on the other side, it does not appear on any of M_ks.

3. $S[i]$ is successfully transmitted and, as $S'[j]$ on the other side, it appears in some M_k although it is matched to another element of S.

The first case happens for at most δn elements as adversary is allowed to delete up to δn many elements.

To analyze the second case, note that the sizes of M_ks descend as k grows since we pick the longest common subsequence in each step. If by the end of this procedure p successfully transmitted symbols are still not matched in any of the matchings, they form a common subsequence of size p between S and the remainder of S'. This leads to the fact that $|M_1| + \cdots + |M_K| \geq K \cdot p$. As $|M_1| + \cdots + |M_K|$ cannot exceed $|S'|$, we have $p \leq |S'|/K$. This bounds above the number of symbols falling into the second category by $|S'|/K$.

Finally, as for the third case, we draw the reader's attention to the fact that each successfully transmitted $S[i]$ which arrives at the other end as $S'[j]$ and mistakenly gets matched to another element of S like $S[k]$ in some M_t, implies that $S[i] = S[k]$. We call the pair (i, k) *a pair of similar elements in S implied by M_t*. Note that there is an actual monotone matching M' from S to S' that corresponds to adversary's actions. As M_t and M' are both monotone, the set of similar pairs in S implied by M_t is a self-matching in S. As stated in Theorem 4.2.3, the number of such pairs cannot exceed $n\varepsilon$. Therefore, there can be at most $n\varepsilon$ successfully transmitted symbols that get mistakenly matched in M_t for any t. Hence, the number of elements falling into the third category is at most $nK\varepsilon$.

Summing up all above-mentioned bounds gives that the number of bad lists can be bounded above by the following.

$$n\delta + \frac{|S'|}{K} + nK\varepsilon \leq n\left(\delta + \frac{1+\gamma}{K} + K\varepsilon\right)$$

This proves the list quality guarantee. As proposed decoding algorithm computes longest common substring K many times between two strings of length n and $(1+\gamma)n$ or less, it will run in $O(K(1+\gamma) \cdot n^2)$ time. □

Theorem 4.3.3. *Suppose that n symbols denoted by $x_1, x_2, \cdots, x_n$ are being communicated through a channel suffering from up to δn deletions for some $0 \leq \delta < 1$ and γn insertions for some constant $\gamma \geq 0$. If one indexes these symbols with an $\varepsilon' = \frac{\varepsilon^2}{4(1+\gamma)}$-synchronization string, then, on the receiving end, it is possible to obtain n lists $A_1, \cdots, A_n$ of size $2(1+\gamma)/\varepsilon$ such that, for at least $n \cdot (1-\delta-\varepsilon)$ of them, $x_i \in A_i$. These lists can be computed in $O\left(n^2(1+\gamma)^2/\varepsilon\right)$ time.*

Proof. Using an $\varepsilon' = \frac{\varepsilon^2}{4(1+\gamma)}$-synchronization string in the statement of Lemma 4.3.2 and choosing $K = \frac{2(1+\gamma)}{\varepsilon}$ directly gives that the runtime is $O\left(n^2(1+\gamma)^2/\varepsilon\right)$ and list hit ratio is at least

$$n \cdot \left(1 - \delta - \frac{1+\gamma}{K} - K \cdot \varepsilon'\right) = n \cdot (1 - \delta - \varepsilon/2 - \varepsilon/2) = n \cdot (1 - \delta - \varepsilon)$$

□

Theorem 4.3.3 facilitates the conversion of list-recoverable error correcting codes into list-decodable insertion-deletion codes as stated in Theorem 4.3.1.

Proof of Theorem 4.3.1. To prove this, we simply index code $\mathcal{C}$ with an $\varepsilon' = \frac{\varepsilon^2}{4(1+\gamma)}$ synchronization string. In the decoding procedure, according to Theorem 4.3.3, the receiver can use the index portion of the received symbol to maintain lists of up to $2(1+\gamma)/\varepsilon \leq l$ candidates for each position of the sent codeword of $\mathcal{C}$ so that $1-\delta-\varepsilon > \alpha$ fraction of those contain the actual corresponding sent message. Having such lists, the receiver can use the decoding function of $\mathcal{C}$ to obtain an L-list-decoding for $\mathcal{C}'$. Finally, the alphabet size and encoding complexity follow from the fact that synchronization strings over alphabets of size $\varepsilon'^{-O(1)}$ can be constructed in linear time (see Chapter 8). □

One can use any list-recoverable error correcting code to obtain insertion-deletion codes according to Theorem 4.3.1. In particular, using the efficient capacity-approaching list-recoverable code introduced by Hemenway, Ron-Zewi, and Wootters [HRZW19], one obtains the insertion-deletion codes as described in Theorem 4.1.1.

Proof of Theorem 4.1.1. By setting parameters $\rho = 1-\delta-\frac{\varepsilon}{2}$, $l = \frac{2(\gamma+1)}{\varepsilon}$, and $\epsilon = \frac{\varepsilon}{4}$ in Theorem 4.2.4, one can obtain a family of codes $\mathcal{C}$ that achieves rate $\rho = 1-\delta-\frac{\varepsilon}{2}$ and is (α, l, L)-recoverable in polynomial time for $\alpha = 1-\delta-\varepsilon/4$ and some $L = \exp(\exp(\exp(\log^* n)))$ (by treating γ and ε as constants). Such family of codes can be found over an alphabet $\Sigma_{\mathcal{C}}$ of size $q = (l/\epsilon)^{O(l/\epsilon^2)} = \left(\frac{\gamma+1}{\varepsilon^2}\right)^{O\left(\frac{\gamma+1}{\varepsilon^3}\right)} = O_{\gamma,\varepsilon}(1)$ or infinitely many integer numbers larger than q.

Plugging this family of codes into the indexing scheme from Theorem 4.3.1 by choosing the parameter $\varepsilon' = \frac{\varepsilon}{4}$, one obtains a family of codes that can recover from $1-\alpha-\varepsilon' = 1-(1-\delta-\varepsilon/4)-\varepsilon/4 = \delta$ fraction of deletions and γ-fraction of insertions and achieves a rate of

$$\frac{1-\delta-\varepsilon/2}{1+\frac{\log|\Sigma_S|}{\log|\Sigma_{\mathcal{C}}|}}$$

which, by taking $|\Sigma_{\mathcal{C}}|$ large enough in terms of ε, is larger than $1-\delta-\varepsilon$. As $\mathcal{C}$ is encodable and decodable in polynomial time, the encoding and decoding complexities of the indexed code will be polynomial as well. □

Remark 4.3.4. *We remark that by using capacity-approaching near-linear-time list-recoverable code introduced in Theorem 7.1 of Hemenway, Ron-Zewi, and Wootters [HRZW19] in the framework of Theorem 4.3.1, one can obtain similar list-decodable insertion-deletion codes as in Theorem 4.1.1 with a randomized quadratic time decoding. Further, one can use the efficient list-recoverable in the recent work of Guruswami and Xing [GX17] to obtain same result as in Theorem 4.1.1 except with polylogarithmic list sizes.*

Chapter 5

Optimally Resilient List-Decodable Synchronization Codes

In this Chapter, we give a complete answer to the following basic question: "What is the maximal fraction of deletions or insertions tolerable by q-ary list-decodable codes with non-vanishing information rate?"

This question has been open even for binary codes, including the restriction to the binary insertion-only setting, where the best-known result was that a $\gamma \leq 0.707$ fraction of insertions is tolerable by some binary code family.

For any desired $\varepsilon > 0$, we construct a family of binary codes of positive rate which can be efficiently list-decoded from any combination of γ fraction of insertions and δ fraction of deletions as long as $\gamma + 2\delta \leq 1 - \varepsilon$. On the other hand, for any γ, δ with $\gamma + 2\delta = 1$ list-decoding is impossible. Our result thus precisely characterizes the feasibility region of binary list-decodable codes for insertions and deletions.

We further generalize our result to codes over any finite alphabet of size q. Surprisingly, our work reveals that the feasibility region for $q > 2$ is *not* the natural generalization of the binary bound above. We provide tight upper and lower bounds that precisely pin down the feasibility region, which turns out to have a $(q-1)$-piece-wise linear boundary whose q corner-points lie on a quadratic curve.

The main technical work in the results of this chapter is proving the existence of code families of sufficiently large *size* with good list-decoding properties for any combination of δ, γ within the claimed feasibility region. We achieve this via an intricate analysis of codes introduced by Bukh and Ma [BM14]. Finally, we give a simple yet powerful concatenation scheme for list-decodable insertion-deletion codes which transforms any such (non-efficient) code family (with vanishing information rate) into an efficiently decodable code family with constant rate.

5.1 Introduction

Error correcting codes have the ability to efficiently correct large fractions of errors while maintaining a large communication rate. The fundamental trade-offs between these two conflicting desiderata have been intensely studied in information and coding theory. Algorithmic coding theory has further studied what trade-offs can be achieved *efficiently*, i.e., with polynomial time encoding and decoding procedures

As stated and discussed earlier on in this thesis, while codes for Hamming errors and the Hamming metric are quite well understood, insdel codes have largely resisted such progress. A striking example of a basic question that is open in the context of synchronization errors is the determination of the maximal fraction of deletions or insertions a unique- or list-decodable binary code with non-vanishing rate can tolerate. That is, we do not even know at what fraction of errors the rate/distance tradeoff for insdel codes hits zero rate. These basic and intriguing questions are open even if one just asks about the existence of codes, irrespective of computational considerations, and even when restricted to the insertion-only setting.

In this chapter, we fully answer these questions for list-decodable binary codes and more generally for codes over any alphabet of a fixed size q. Our results are efficient and work for any combination of insertions and deletions from which list decoding is information-theoretically feasible at all.

5.1.1 Prior Results and Related Works

We now give an overview of the previous works related to the main thrust of this chapter, namely the maximal tolerable fraction of worst-cast deletions or insertions for unique- and list-decodable code families with non-vanishing rate.

Unique Decoding. Let us first review the situation for unique decoding, where the decoder must determine the original transmitted codeword. For unique decoding of binary codes, the maximal tolerable fraction of deletions is easily seen to be at most $\frac{1}{2}$ because otherwise either all zeros or all ones in a transmitted codeword can be deleted. (For q-ary codes, this fraction becomes $1 - 1/q$.) On the other hand, for a long time the best (existential) possibility results for unique-decodable binary codes stemmed from analyzing random binary codes.

In the Hamming setting, random codes often achieve the best known parameters and trade-offs, and a lot of effort then goes into finding efficient constructions and decoding algorithms for codes that attempt to come close to the random constructions. However, the edit distance is combinatorially intricate and even analyzing the expected edit distance of two random strings, which is the first step in analyzing random codes, is highly non-trivial.

Lueker [Lue09], improving upon earlier results by Dančík and Paterson [Dan94, DP95], proved that the expected fractional length of the longest common subsequence between two random strings lies between 0.788071 and 0.826280 (the exact value is still unknown). Using this, one can show that a random binary code of positive rate can tolerate between 0.23 and 0.18 fraction of deletions or insertions. Edit distance of random q-ary strings were studied by Kiwi, Loebl, and Matoušek [KLM05], leading to positive rate random

codes by Guruswami and Wang [GW17] that correct $1 - \Theta(\frac{1}{\sqrt{q}})$ fraction of deletions for asymptotically large q. Because random codes do not have efficient decoding and encoding procedures these results were purely existential. Computationally efficient binary codes of non-vanishing rate tolerating some small unspecified constant fraction of insertions and deletions were given by Schulman and Zuckerman [SZ99]. Guruswami and Wang [GW17] gave binary codes that could correct a small constant fraction of deletions with rate approaching 1, and this was later extended to handle insertions as well [GL16].

In the regime of low-rate and large fraction of deletions, Bukh and Guruswami [BG16] gave a q-ary code construction that could tolerate up to a $\frac{q-1}{q+1}$ fraction of deletions, which is $\frac{1}{3}$ for binary codes. Note that this beats the performance of random codes. Together with Håstad [BGH17] they later improved the deletion fraction to $1 - \frac{2}{q+\sqrt{q}}$ or $\sqrt{2} - 1 \approx 0.414$ for binary codes. This remains the best known result for unique-decodable codes and determining whether there exist binary codes capable of correcting a fraction of deletions approaching $\frac{1}{2}$ remains a fascinating open question.

List decoding. The situation for list-decodable codes over small alphabets is equally intriguing. In list-decoding, one relaxes the decoding requirement from having to output the codeword that was sent to having to produce a (polynomially) small list of codewords which includes the correct one. The trivial limit of 1/2 fraction deletions for unique-decoding binary codes applies equally well for list-decoding. In their paper, Guruswami and Wang [GW17] showed that this limit can be approached by efficiently list-decodable binary codes. Similarly, q-ary codes list-decodable from a deletion fraction approaching the optimal $1 - 1/q$ bound can be constructed.

However, the situation was not well understood when insertions are also allowed. It had already been observed by Levenshtein [Lev65] that (at least existentially) insertions and deletions are equally hard to correct for unique-decoding, in that if a code can correct t deletions then it can also correct any combination of t insertions and deletions. This turns out to be not true for list-decoding. This was demonstrated pointedly in Chapter 4, where it is shown that arbitrary large $\gamma = O(1)$ fractions of insertions (possibly exceeding 1) can be tolerated by list-decodable codes over sufficiently large constant alphabets (see Theorem 4.1.1), whereas the fraction of deletions δ is clearly bounded by 1. Indeed, the fraction of insertions γ does not even factor into the rate of these list-decodable insertion-deletion codes—this rate can approach the optimal bound of $1 - \delta$ where δ is the deletion fraction. The result in Chapter 4, however, applies only to sufficiently large constant alphabet sizes, and does not shed any light on the list-decodability of *binary* (or any fixed alphabet) insdel codes.

Considering a combination of insertions and deletions, the following bound is not hard to establish.

Proposition 5.1.1. *For any integer q and any $\delta, \gamma \geq 0$ with $\frac{\delta}{1-\frac{1}{q}} + \frac{\gamma}{q-1} \geq 1$ there is no family of constant rate codes of length n which are list-decodable from δn deletions and γn insertions.*

For the case of insertion-only binary codes, the above limits the maximum fraction of insertions to 100%, which is twice as large as the best possible deletion fraction of 1/2.

Turning to existence/constructions of list-decodable codes for insertions, recall that the codes of Bukh, Guruswami, Håstad (BGH) could unique-decode (and thus also list-decode) a fraction of 0.414 insertions (indeed any combination of insertions and deletions totaling 0.414 fraction). Wachter-Zeh [WZ18] recently put forward a Johnson-type bound for insdel codes. The classical Johnson bound works in the Hamming metric, and connects unique-decoding to list-decoding (for Hamming errors) by showing that any unique-decodable code must also be list-decodable from an even larger fraction of corruptions. One intriguing implication of Wachter-Zeh's Johnson bound for insdel codes is that any unique-decodable insdel code which tolerates a $\frac{1}{2}$ fraction of deletions (or insertions) would automatically also have to be (existentially) list-decodable from a 100% fraction of insertions. Therefore, even if one is interested in unique-decoding, e.g., closing the above-mentioned gap between $\sqrt{2}-1$ and $\frac{1}{2}$, this establishes the search for maximally list-decodable binary codes from insertions as a good and indeed necessary step towards this goal. On the other hand, proving any non-trivial impossibility result bounding the maximal fraction of insertions of list-decodable binary codes away from 100% would directly imply an impossibility result for unique-decoding binary codes from a deletion fraction approaching $\frac{1}{2}$.

Follow-up work by Hayashi and Yasunaga [HY18] corrected some subtle but crucial bugs in [WZ18] and reproved a corrected Johnson Bound for insdel codes. They furthermore showed that the BGH codes [BGH17] could be list-decoded from a fraction ≈ 0.707 of insertions. Lastly, via a concatenation scheme used in [GW17, GL16] they furthermore made these codes efficient. In summary, for the binary insertion-only setting, the largest fraction of insertions that we knew to be list-decodable (even non-constructively) was ≈ 0.707.

5.1.2 Our Results

We close the above gap and show binary codes which can be list-decoded from a fraction $1-\varepsilon$ fraction of insertions, for any desired constant $\varepsilon > 0$. In fact, we give a single family of codes that are list-decodable from any mixed combination of γ fraction of insertions and δ fraction of deletions, as long as $2\delta + \gamma \leq 1 - \varepsilon$.

Theorem 5.1.2. *For any $\varepsilon \in (0,1)$ and sufficiently large n, there exists a constant rate family of efficient binary codes that are L-list decodable from any δn deletions and γn insertions in $\mathsf{poly}(n)$ time as long as $\gamma + 2\delta \leq 1 - \varepsilon$ where n denotes the block length of the code, $L = O_\varepsilon(\exp(\exp(\exp(\log^* n))))$, and the code achieves a rate of $\exp\left(-\frac{1}{\varepsilon^{10}}\log^2\frac{1}{\varepsilon}\right)$.*

Since the computationally efficient codes from Theorem 5.1.2 match the bounds from Proposition 5.1.1 for every δ, γ, this nails down the entire feasibility region for list-decodability from insertions and deletions for the binary case. We stress that while we get constructive results, even the existence of inefficiently list-decodable codes, that too just for the insertion-only setting, was not known prior to this work.

In the above result, the rather weird looking bound on the list-size is inherited from results on list-decoding from a huge number insertions over larger alphabets Chapter 4, which in turn is inherited from the list-size bounds for the list-recoverable algebraic-geometric code constructions in [GX13].

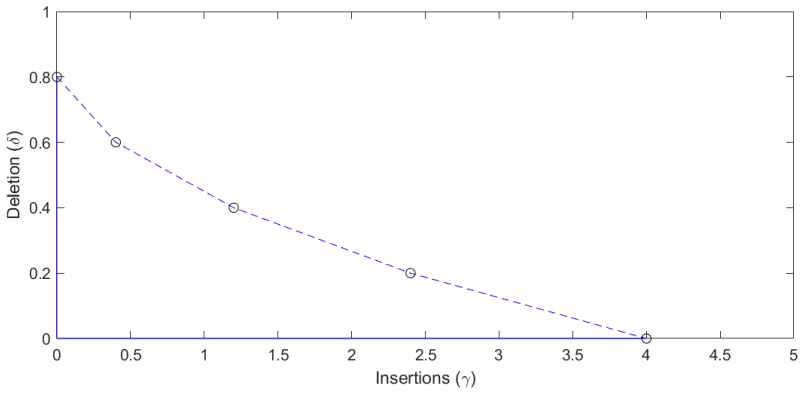


Figure 5.1: Feasibility region for $q = 5$.

We use similar construction techniques to obtain codes with positive rate over any arbitrary alphabet size q that are list-decodable from any fraction of insertions and deletions under which list-decoding is possible. We, thus, precisely identify the feasibility region for any alphabet size, together with an efficient construction. Again, recall that the existence of such codes was not known earlier, even for the insertion-only case.

Theorem 5.1.3. *For any positive integer $q \geq 2$, define F_q as the concave polygon defined over vertices $\left(\frac{i(i-1)}{q}, \frac{q-i}{q}\right)$ for $i = 1, \cdots, q$ and $(0, 0)$. (An illustration for $q = 5$ is presented in Fig. 5.1). F_q does not include the border except the two segments $[(0, 0), (q - 1, 0))$ and $[(0, 0), (0, 1 - 1/q))$. Then, for any $\varepsilon > 0$ and sufficiently large n, there exists a family of q-ary codes that, as long as $(\gamma, \delta) \in (1 - \varepsilon)F_q$, are efficiently L-list decodable from any δn deletions and γn insertions where n denotes the block length of the code, $L = O(\exp(\exp(\exp(\log^* n))))$, and the code achieves a positive rate of $\exp\left(-\frac{1}{\varepsilon^{10}} \log^2 \frac{1}{\varepsilon}\right)$.*

We further show in Section 5.5 that for any pair of positive real numbers $(\gamma, \delta) \notin F_q$, there exists no infinite family of q-ary codes with rate bounded away from zero that can be list decoded from a δ-fraction of deletions plus a γ-fraction of insertions.

5.1.3 Our Techniques

We achieve these results using two ingredients, each interesting in its own right. The first is a simple new concatenation scheme for list-decodable insdel codes which can be used to boost the rate of insdel codes. The second component, which constitutes the bulk of this work, is a technically intricate proof of the list-decoding properties of the Bukh-Ma codes [BM14] which have good (edit) distance properties but a tiny sub-constant rate. We note that these codes were the inner codes in the "clean construction" in the BGH work on codes unique-decodable from a 1/3 insdel fraction [BGH17]. This was driven by a property of these codes called the *span*, which is a stronger form of edit distance that applies at all scales. The Bukh-Ma codes were also used by Guruswami and Li [GL20] in their existence proof of codes of positive rate for correcting a fraction of *oblivious deletions* approaching 1. In this work, the non-trivial list-decodability property of the Bukh-Ma codes drives our result.

Concatenating List-Decodable Insdel Codes

Our first ingredient is a simple but powerful framework for constructing list-decodable insertion-deletion codes via code concatenation. Recall that code concatenation which composes the encoding of an *outer* code C_{out} with an *inner* code C_{in} whose size equals the alphabet size of C_{out}.

In our approach, the outer code C_{out} is chosen to be a list-decodable insdel code C_{out} over an alphabet that is some large function of $1/\varepsilon$, but which has constant rate and is capable of tolerating a huge number of insertions. The inner code C_{in} is chosen to be a list-decodable insdel code over a fixed alphabet of the desired size q, which has non-trivial list decoding properties for the desired fraction δ, γ of deletions and insertions.

We show that even if C_{in} has an essentially arbitrarily bad sub-constant rate and is not efficient, the resulting q-ary insdel code does have constant rate, and can also be efficiently list decoded from the same fraction of insertions and deletions as C_{in}. For the problem considered in this chapter, this framework essentially provides efficiency of codes for free. More importantly, it reduces the problem of finding good *constant-rate* insdel codes over a fixed alphabet to finding a family of good list-decodable insdel codes *with an arbitrarily large number of codewords*, and a list-size bounded by some fixed function of $1/\varepsilon$.

Our decoding procedure for concatenated list-decodable insdel codes is considerably simpler than similar schemes introduced in earlier works [GW17, GL16, BGH17, SZ99]. Of course, the encoding is simply given by the standard concatenation procedure. The decoding is done by (i) list-decoding shifted intervals of the received string using the inner code C_{in}, (ii) creating a single string from the symbols in these lists, and (iii) using the list-decoding algorithm of the outer code on this string (viewed as a version of the outer codeword with some number of deletions and insertions).

The main driving force behind why this simplistic sounding approach actually works is a judicious choice of the outer code C_{out}. Specifically, we use the codes from Chapter 4 which can tolerate a very large number of insertions. This means that the many extra symbols coming from the list-decodings of the inner code C_{in} and the choice of overlapping intervals does not disrupt the decoding of the outer code.

5.1.4 Analyzing the List-Decoding Properties of Bukh-Ma Codes

The main technical challenge that remains is to construct or prove the existence of arbitrarily large binary codes with optimal list decoding properties for any γ, δ (and q). For this we turn to a simple family of codes introduced by Bukh and Ma [BM14], which consist of strings $(0^r\ 1^r)^{\frac{n}{r}}$ which oscillate between 0's and 1's with different frequencies. (Below we will refer to r as the *period*, and $1/r$ should be thought of as the *frequency* of alternation.)

A simple argument shows that the edit distance between any two such strings with sufficiently different periods is maximal, resulting in a tolerable fraction of edit errors of $\frac{1}{2}$ for unique decoding. The Johnson bound of [WZ18, HY18] implies that this code must also be list-decodable from a full fraction 100% of insertions. Therefore, using these codes as the inner codes in the above-mentioned concatenation scheme resolves the list-decoding question for the insertion-only setting. (The deletion-only setting is oddly easier as just

random inner codes suffice, and was already resolved in [GW17].) This also raises hope that the Bukh-Ma codes might have good list-decoding properties for other γ, δ as well. Fortunately, this turns out to be true, though establishing this involves an intricate analysis that constitutes the bulk of the technical work in this chapter.

Theorem 5.1.4. *For any $\varepsilon > 0$ and sufficiently large n, let $C_{n,\varepsilon}$ be the following Bukh-Ma code:*

$$C_{n,\varepsilon} = \left\{ (0^r 1^r)^{\frac{n}{2r}} \,\middle|\, r = \left(\frac{1}{\varepsilon^4}\right)^k, k < \log_{1/\varepsilon^4} n \right\}.$$

For any $\delta, \gamma \geq 0$ where $\gamma + 2\delta < 1 - \varepsilon$, $C_{n,\varepsilon}$ is list-decodable from any δn deletions and γn insertions with a list size of $O\left(\frac{1}{\varepsilon^3}\right)$.

In order to prove Theorem 5.1.4 we first introduce a new correlation measure which expresses how close a string is to any given frequency (or Bukh-Ma codeword) if one allows for both insertions and deletions each weighted appropriately. Using this we want to show that it is impossible to have a single string v which is more than ε-correlated with more than $\Theta_\varepsilon(1)$ frequencies.

Intuitively, one might expect that each correlation can be (fractionally) attributed to a (disjoint) part of v which would result in the maximum number of ε-close frequencies to be at most $1/\varepsilon$. This, however, turned out to be false. Instead, we use a proof technique which is somewhat reminiscent of the one used to establish the polarization of the martingale of entropies in the analysis of polar codes [Ari09, BGN$^+$18].

In more detail, we think of recursively sub-sampling smaller and smaller nested substrings of v, and analyze the expectation and variance of the bias between the fraction of 0's and 1's in these substrings. More precisely, we order the run lengths $r_1, r_2, \ldots$ that are ε-correlated with v in decreasing order and first sample a substring v_1 with $r_1 \gg |v_1| \gg r_2$ from v. While the expected zero-one bias in v_1 is the same as in v, we show that the variance of this bias is an increasing function in the correlation with $(0^{r_1}1^{r_1})^{\frac{n}{2r_1}}$. Intuitively, v_1 cannot be too uniform on an scale of length l if it is correlated with r_1.

Put differently, in expectation the sampled substring v_1 will land in a part of v which is either (slightly) correlated to one of the long stretches of zeros in v or in a part which is correlated with a long stretch of ones in v, resulting in at least some variance in the bias of v_1. Because the scales $r_2, r_3, \ldots$ are so much smaller than v_1, this sub-sampling of v_1 furthermore preserves the correlation with these scales intact, at least in expectation.

Next we sample a substring v_2 with $r_2 \gg |v_2| \gg r_3$ within v_1. Again, the bias in v_2 stays the same as the one in v_1 in expectation but the sub-sampling introduces even more variance given that v_1 is still non-trivially correlated with the string with period r_2. The evolution of the bias of the strings $v_1, v_2, \ldots$ produced by this nested sampling procedure can now be seen as a martingale with the same expectation but an ever increasing variance. Given that the bias is bounded in magnitude by 1, the increase in variance cannot continue indefinitely. This limits the number of frequencies a string v can be non-trivially correlated with, which is exactly what we were after.

Our generalization to larger q-ary alphabets follows the same high level blueprint, but is technically even more delicate. Recall that in the non-binary case, there are $(q-1)$ different linear trade-offs between δ, γ depending on the exact regime they lie in.

5.2 Preliminaries

5.2.1 List-Decodable Insertion-Deletion Codes

For the sake of convenience, we restate the following codes from Chapter 4 which will be used as the outer code in our constructions.

Theorem 5.2.1 (Restatement of Theorem 4.1.1)**.** *For every $\delta, \varepsilon \in (0,1)$ and constant $\gamma > 0$, there exist a family of list-decodable insdel codes that can protect against δ-fraction of deletions and γ-fraction of insertions and achieves a rate of $1 - \delta - \varepsilon$ or more over an alphabet of size $\left(\frac{\gamma+1}{\varepsilon^2}\right)^{O\left(\frac{\gamma+1}{\varepsilon^3}\right)} = O_{\gamma,\varepsilon}(1)$. These codes are list-decodable with lists of size $L_{\varepsilon,\gamma}(n) = \exp\left(\exp\left(\exp\left(\log^* n\right)\right)\right)$, and have polynomial time encoding and decoding complexities.*

5.2.2 Strings, Insertions and Deletions, and Distances

In this section, we provide preliminary definitions on strings, edit operations, and related notions. We start by definition of count and bias.

Definition 5.2.2 (Count and Bias)**.** *We define $\mathsf{count}_a(w) = |\{i | w[i] = a\}|$ as the number of appearances of symbol a in string w. The bias of a binary string w is the normalized difference between the appearances of zeros and ones in w, i.e., $\mathsf{bias}(w) = \frac{\mathsf{count}_1(w) - \mathsf{count}_0(w)}{|w|}$. With this definition, $\mathsf{count}_0(w) = \frac{1-\mathsf{bias}(w)}{2}|w|$ and $\mathsf{count}_1(w) = \frac{1+\mathsf{bias}(w)}{2}|w|$.*

Next, we formally define a *matching* between two strings.

Definition 5.2.3 (Matching)**.** *A matching M of size k between two strings S and S' is defined to be two sequences of k integer positions $0 < i_1 < \ldots < i_k \leq |S|$ and $0 < i'_1 < \ldots < i'_k \leq |S'|$ for which $S[i_j] = S'[i'_j]$ for all $j \leq k$. The subsequence induced by a matching M is simply $S[i_1], \ldots, S[i_k]$. Every common subsequence between S and S' implicitly corresponds to a matching and we use the two interchangeably.*

We now proceed to define the important notion of advantage.

Definition 5.2.4 (Advantage of a Matching)**.** *Let M be a matching between two binary strings a and b. The* advantage of the matching *M is defined as*

$$\mathsf{adv}_M = \frac{3|M| - |a| - |b|}{|a|}.$$

Definition 5.2.5 (Advantage)**.** *For a given pair of strings a and b, the* advantage of *a* to *b is defined as the advantage of the matching M that corresponds to the largest common subsequence between them, i.e., $\mathsf{adv}(a,b) = \mathsf{adv}_{M=\mathrm{LCS}(a,b)}$. It is easy to verify that the longest common subsequence M maximizes the advantage among all matchings from a to b.*

We now make the following remark that justifies the notion of advantage as defined above. Note that any matching between two strings a and b implies a set of insertions and deletions to convert b to a which is, to delete all unmatched symbols in b and insert all unmatched symbols in a within the remaining symbols.

Remark 5.2.6. *Consider strings a and b and matching M between them. Think of a as a distorted version of b and let δ_M and γ_M represent the fraction of deletions and insertions needed to convert b to a as suggested by M, i.e.,*

$$\delta_M = \frac{\text{Number of unmatched symbols in } b}{|b|} = \frac{|b| - |M|}{|b|},$$

and

$$\gamma_M = \frac{\text{Number of unmatched symbols in } a}{|b|} = \frac{|a| - |M|}{|b|}.$$

The adv_M function tracks the value of $|b|(1 - 2\delta_M - \gamma_M)$ normalized by $|a|$ rather than $|b|$.

$$\mathsf{adv}_M(a, b) = \frac{3|M| - |a| - |b|}{|a|} = \frac{3|b|(1 - \delta_M) - |b|(1 - \delta_M + \gamma_M) - |b|}{|a|} = \frac{|b|}{|a|} \cdot (1 - 2\delta_M - \gamma_M)$$

We will make use of this unnatural normalization later on.

We now extend the definition of advantage to the case where the second argument is an infinite string.

Definition 5.2.7 (Infinite Advantage)**.** *For a finite string a and infinite string b, the advantage of a to b is defined as the minimum advantage that a has over all substrings of b.*

$$\mathsf{adv}(a, b) = \min_{b' = b[i,j]} \mathsf{adv}(a, b').$$

We now define a family of binary strings called *Alternating Strings*.

Definition 5.2.8 (Alternating Strings)**.** *For any positive integer r, we define the infinite alternating string of run-length r as $A_r = (0^r 1^r)^\infty$ and denote its prefix of length l with $A_{r,l} = A_r[1, l]$.*

We finish the preliminaries by the following lemma stating some properties of the notions defined through this section.

Lemma 5.2.9. *The following properties hold true:*

- *For any pair of binary strings S_1, S_2 where $\mathsf{adv}(S_1, S_2) > 0$, lengths of S_1 and S_2 are within a factor of two of each other, i.e, $\min(|S_1|, |S_2|) \geq \frac{\max(|S_1|, |S_2|)}{2}$.*
- *For any binary string S and integer r, $\mathsf{adv}(S, A_r) \geq -\frac{1}{2}$*

Proof. For the first part, let $M = \mathrm{LCS}(S_1, S_2)$. We have that $\mathsf{adv}(S_1, S_2) \geq 0 \Rightarrow 3|M| \geq |S_1| + |S_2|$, which, as $|M| \leq \min(|S_1|, |S_2|)$, implies that $\min(|S_1|, |S_2|) \geq \frac{\max(|S_1|, |S_2|)}{2}$.

For the second part, let $n = |S|$ and assume that $b \in \{0, 1\}$ is the most frequent bit in S and there are m occurrences of b in S. Take a substring S' in A_r as the smallest string that starts at the beginning of a b^r block and contains the same number of bs as S. The size of S' is no more than $2m$ and the longest common subsequence between S and S' is at least m. Therefore,

$$\mathsf{adv}(S, A_r) \geq \mathsf{adv}(S, S') \geq \frac{3|M| - |S| - |S'|}{|S|} \geq \frac{3m - 2m - 2m}{n} \geq \frac{-m}{n} \geq -\frac{1}{2}. \qquad \square$$

5.3 Proof of Theorem 5.1.4: List-Decoding for Bukh-Ma Codes

To prove this theorem, we assume for the sake of contradiction that there exists a string v and $k > \frac{1200}{\varepsilon^3}$ members of $C_{n,\varepsilon}$ like $A_{r_1,n}, A_{r_2,n}, \cdots, A_{r_k,n}$, so that each $A_{r_i,n}$ can be converted to v with I_i insertions and D_i deletions where $I_i + 2D_i \leq n(1 - \varepsilon)$. We define the indices in a way that $r_1 > r_2 > \cdots > r_k$. Given the definition of $C_{n,\varepsilon}$, $r_i \geq \frac{r_{i+1}}{\varepsilon^4}$. We first show that, for all $i = 1, 2, \cdots, k$, $\mathsf{adv}(v, A_{r_i,n}) \geq \frac{\varepsilon}{2}$.

Lemma 5.3.1. *For any $1 \leq i \leq k$, $\mathsf{adv}(v, A_{r_i,n}) \geq \frac{\varepsilon}{2}$.*

Proof. Let M_i denotes the matching that corresponds to the set of I_i insertions and D_i deletions that convert $A_{r_i,n}$ to v.

$$I_i + 2D_i \leq n(1 - \varepsilon) \Rightarrow n - I_i - 2D_i \geq n\varepsilon \Rightarrow 1 - \gamma_i - 2\delta_i \geq \varepsilon$$

Note that according to Remark 5.2.6, $\mathsf{adv}(v, A_{r_i,n}) = \frac{n}{|v|} \cdot (1 - \gamma_i - 2\delta_i)$. Thus, $\mathsf{adv}(v, A_{r_i,n}) \geq \frac{n}{|v|}\varepsilon \geq \frac{\varepsilon}{2}$. The last step follows from the first item of Lemma 5.2.9. $\square$

Having Lemma 5.3.1, we are ready to prove Theorem 5.1.4. We start with defining a couple of sequences of random variables via random sampling of nested substrings of v. We split the string v into substrings of size $l_1 = r_1\varepsilon^2$, pick one uniformly at random and denote it by v_1. We define random variable $A_1 = \mathsf{adv}(v_1, A_{r_1})$ and random variable $B_1 = \mathsf{bias}(v_1)$. Similarly, we split v_1 into substrings of length $l_2 = r_2\varepsilon^2$ and pick v_2 uniformly at random and define $A_2 = \mathsf{adv}(v_2, A_{r_2})$ and $B_2 = \mathsf{bias}(v_2)$. Continuing this procedure, one can obtain the two sequences of random variables $A_1, A_2, \cdots, A_k$ and $B_1, B_2, \cdots, B_k$. We will prove the following.

Lemma 5.3.2. *The following hold for $A_1, A_2, \cdots, A_k$ and $B_1, B_2, \cdots, B_k$.*

1. $\mathbb{E}[B_i] = \mathsf{bias}(v)$
2. $\mathbb{E}[A_i] \geq \frac{\varepsilon}{2}$

Proof. Note that one can think of v_i as a substring of v that is obtained by splitting v into substrings of length l_i and choosing one uniformly at random. Let U denote the set of all such substrings. We have that

$$\begin{aligned}\mathbb{E}[B_i] &= \sum_{\hat{v}\in U}\frac{1}{|U|}\cdot \mathsf{bias}(\hat{v}) = \frac{1}{|U|}\sum_{\hat{v}\in U}\frac{\mathsf{count}_1(\hat{v})-\mathsf{count}_0(\hat{v})}{l_i}\\ &= \frac{\mathsf{count}_1(v)-\mathsf{count}_0(v)}{|U|\cdot l_i} = \mathsf{bias}(v).\end{aligned}$$

A similar argument proves the second item. Take the matching M_i between v and $A_{r_i,n}$ that achieves the advantage $\mathsf{adv}(v, A_{r_i,n})$, i.e., the largest matching between v and $A_{r_i,n}$. Take some $\hat{v} \in U$; $\hat{v}$ is mapped to some substring in $A_{r_i,n}$ under M_i. We call that substring of $\hat{v}$, *the projection of* $\hat{v}$ *under* M_i and denote it by $\hat{v} \to M_i$. We also represent the subset of M_i that appears between $\hat{v}$ and $\hat{v} \to M_i$ with $M_i[\hat{v}]$.

For a $\hat{v} \in U$, we define $a(\hat{v})$ as the value for advantage that is yielded by the matching $M_i[\hat{v}]$ between $\hat{v}$ and $\hat{v} \to M_i$. In other words, $a(\hat{v}) = \frac{3|M_i[\hat{v}]|-|\hat{v}|-|\hat{v}\to M_i|}{|\hat{v}|}$. Given the definitions of advantage and infinite advantage, we have that

$$a(\hat{v}) \le \mathsf{adv}(\hat{v}, \hat{v}\to M_i) \le \mathsf{adv}(\hat{v}, A_{r_i}).$$

This can be used to prove the second item as follows:

$$\begin{aligned}\mathbb{E}[A_i] &= \sum_{\hat{v}\in U}\frac{1}{|U|}\cdot\mathsf{adv}(\hat{v}, A_{r_i}) \ge \frac{1}{|U|}\cdot\sum_{\hat{v}\in U} a(\hat{v})\\ &= \frac{1}{|U|}\cdot\sum_{\hat{v}\in U}\frac{3|M_i[\hat{v}]|-|\hat{v}|-|\hat{v}\to M_i|}{|\hat{v}|} = \frac{1}{|U|\cdot|\hat{v}|}\cdot\sum_{\hat{v}\in U}(3|M_i[\hat{v}]|-|\hat{v}|-|\hat{v}\to M_i|)\\ &= \frac{1}{|v|}\cdot(3|M_i|-|v|-|A_{r_i,n}|) = \mathsf{adv}(v, A_{r_i,n}) \ge \frac{\varepsilon}{2}\end{aligned}$$

where the last step follows from Lemma 5.3.1. □

Lemma 5.3.3. *For the sequence $B_1, B_2, \cdots, B_k$, we have*

$$\mathrm{Var}(B_{i+1}) \ge \mathrm{Var}(B_i) + \frac{\varepsilon^3}{1200}, \quad \forall 1 \le i < k.$$

Proof. To analyze the relation of $\mathrm{Var}(B_i)$ and $\mathrm{Var}(B_{i+1})$, we use the law of total variance and condition the variance of B_{i+1} on v_i, i.e., the substring chosen in the ith step of the stochastic process, from which we sub sample v_{i+1}.

$$\begin{aligned}\mathrm{Var}(B_{i+1}) &= \mathrm{Var}\left(\mathbb{E}[B_{i+1}|v_i]\right) + \mathbb{E}\left[\mathrm{Var}(B_{i+1}|v_i)\right]\\ &= \mathrm{Var}\left(B_i\right) + \mathbb{E}\left[\mathrm{Var}(B_{i+1}|v_i)\right] \qquad (5.1)\end{aligned}$$

Equation (5.1) comes from the fact that the average bias of substrings of length l_{i+1} in v_i is equal to the bias of v_i. Having this, we see that it suffices to show that $\mathbb{E}\left[\mathrm{Var}(B_{i+1}|v_i)\right] \ge$

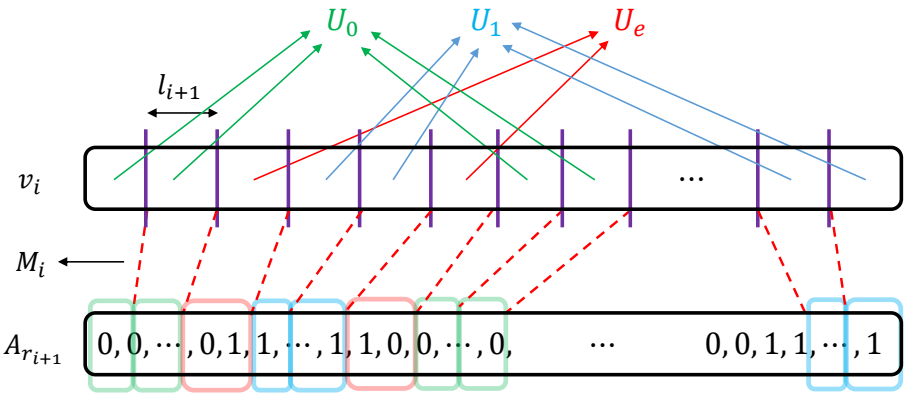


Figure 5.2: Partitioning substrings of length l_{i+1} into three sets U_0, U_1, U_e

$\varepsilon^3/1200$. We remind the reader that v_{i+1} is obtained by splitting v_i into substrings of length $l_{i+1} = r_{i+1}\varepsilon^2$ and choosing one at random. We denote the set of such substrings by U. Also, there is a matching M_i between v_i and $A_{r_{i+1}}$ with advantage ε or more. Any substring of length l_{i+1} is mapped to some substring in $A_{r_{i+1}}$, i.e., its projection of the substring under M_i. Note there are three different possibilities for such projection. It is either an all zeros string, an all one string, or a string that contains both zeros and ones. We partition U into three sets U_0, U_1, and U_e based on which case the projection belongs to. (See Fig. 5.2)

We partition the sample space into three events E_0, E_1, and E_e based on whether v_{i+1} belongs to U_0, U_1, or U_e respectively. We also define the random variable T over $\{0, 1, e\}$ that indicates which one of E_0, E_1, or E_e happens. Once again, we use the law of total variance to bound $\mathbb{E}[\mathrm{Var}(B_{i+1}|v_i)]$.

$$\begin{aligned}\mathbb{E}[\mathrm{Var}(B_{i+1}|v_i)] &= \mathbb{E}_{v_i}\left[\mathrm{Var}_T(\mathbb{E}[B_{i+1}|v_i, T]) + \mathbb{E}_T[\mathrm{Var}(B_{i+1}|v_i, T)]\right] \\ &\geq \mathbb{E}_{v_i}\left[\mathrm{Var}_T(\mathbb{E}[B_{i+1}|v_i, T])\right] \qquad (5.2)\end{aligned}$$

Note that the term $\mathrm{Var}_T(\mathbb{E}[B_{i+1}|v_i, T])$ refers to variance of a 3-valued random variable that takes the value $\mathbb{E}_{v_i}[B_{i+1}|v_i, T = t]$ with probability $\Pr\{T = t|v_i\}$ for $t \in \{0, 1, e\}$. We use three important facts about this distribution to bound its variance from below.

First, $\Pr\{T = e|v_i\} \leq 2\varepsilon^2$. To see this, note that the run length of $A_{r_{i+1}}$ is $r_{i+1} = \frac{l_{i+1}}{\varepsilon^2}$ and the length of the projection of v_i in A_{r_i} under the matching that yields the optimal $\mathsf{adv}(v_i, A_{r_i})$ is no more than $2|v_i| = 2l_i$ (See Lemma 5.2.9). Therefore, $|U_e| \leq \frac{2l_i}{r_{i+1}}$ and consequently no more that a $\frac{2l_i/r_{i+1}}{l_i/l_{i+1}} = 2\varepsilon^2$ fraction of strings in U might be mapped to a substring of $A_{r_{i+1}}$ that crosses the border of some $0^{r_{i+1}}$ and $1^{r_{i+1}}$ intervals.

Secondly, for any $j \in \{0, 1\}$, $\Pr\{T = j|v_i\} \geq \frac{\mathsf{adv}(v_i, A_{r_{i+1}}) - 8\varepsilon^2}{8}$. This can be showed as follows. Let M_i^j represent the subset of pairs of M_i with one end in U_j for $j \in \{0, 1, e\}$ and $v_i \to M_i$ represent the substring of $A_{r_{i+1}}$ where v_i is projected under M_i. Note that $\Pr\{T = j|v_i\} = \frac{|U_j|}{|U|} = \frac{|U_j| \cdot l_i}{|v_i|} \geq \frac{|M_i^j|}{|v_i|} \geq \frac{|M_i^j|}{2|v_i \to M_i|}$. Assume for contradiction that $\Pr\{T = j|v_i\} < \frac{\mathsf{adv}(v_i, A_{r_{i+1}}) - 8\varepsilon^2}{8}$ for some j. Then, $|M_i^j| < |v_i \to M_i| \frac{\mathsf{adv}(v_i, A_{r_{i+1}}) - 8\varepsilon^2}{4}$, which

since $|M_i^{j'}| \leq \frac{|v_i \to M_i|}{2}$ for $j' \in \{0,1\}$ and $|M_i^e| \leq 2\varepsilon^2 |v_i \to M_i|$, gives that $|M_i| < |v_i \to M_i| \left(\frac{1}{2} + 2\varepsilon^2 + \frac{\mathsf{adv}(v_i, A_{r_{i+1}}) - 8\varepsilon^2}{4}\right) = |v_i \to M_i| \left(\frac{1}{2} + \frac{\mathsf{adv}(v_i, A_{r_{i+1}})}{4}\right)$. However,

$$\mathsf{adv}_{M_i} = \frac{3|M_i| - |v_i| - |p|}{|v_i|} \Rightarrow 2|M_i| - |p| \geq |v_i| \mathsf{adv}_{M_i} \Rightarrow |M_i| \geq |p| \left(\frac{1}{2} + \frac{\mathsf{adv}_{M_i}}{4}\right).$$

This contradiction implies that $\Pr\{T = j | v_i\} \geq \frac{\mathsf{adv}(v_i, A_{r_{i+1}}) - 8\varepsilon^2}{8}$.

The third and final important ingredient is provided by the following lemma that we prove later on.

Lemma 5.3.4. *The following holds true:*

$$\Big| \mathbb{E}\left[B_{i+1} | v_i, T = 0\right] - \mathbb{E}\left[B_{i+1} | v_i, T = 1\right] \Big| \geq \frac{\mathsf{adv}(v_i, A_{r_{i+1}}) - 5\varepsilon^2}{3}$$

To summarize, the above three properties imply that we have a three-valued random variable where the probability for one value is minuscule and there is at least $[\mathsf{adv}(v_i, A_{r_{i+1}}) - 5\varepsilon^2]/3$ difference between the other two values each occurring with adequately large probabilities. This is enough for us to bound below the variance of such random variable. The following straightforward lemma abstracts this.

Lemma 5.3.5. *Let X be a random variable that can take values a_0, a_1, and a_2 where $\Pr\{X = a_i\} \geq \xi$ for $i \in \{0,1\}$. Then, we have that $\mathrm{Var}(X) \geq \frac{\xi}{2}(a_0 - a_1)^2$.*

Proof. $\mathrm{Var}(X) = \sum_{a_i} \Pr\{X = a_i\}(a_i - \bar{X})^2 \geq \xi \left[(a_0 - \bar{X})^2 + (a_1 - \bar{X})^2\right] \geq \frac{\xi}{2}(a_0 - a_1)^2$. □

Applying Lemma 5.3.5 to our random variable gives that:

$$\mathrm{Var}_T(\mathbb{E}\left[B_{i+1} | v_i, T\right]) \geq \frac{1}{144} \left(\mathsf{adv}(v_i, A_{r_{i+1}}) - 8\varepsilon^2\right) \left(\mathsf{adv}(v_i, A_{r_{i+1}}) - 5\varepsilon^2\right)^2$$

Note the right hand side of this inequality is negative when $\mathsf{adv}(v_i, A_{r_{i+1}}) \leq 8\varepsilon^2$. Therefore, we define function $g(x)$ as a function that takes value of $\frac{(x - 8\varepsilon^2)(x - 5\varepsilon^2)}{144}$ when $x > 8\varepsilon^2$ and zero otherwise. Note that g is a convex function. We have that

$$\mathrm{Var}_T(\mathbb{E}\left[B_{i+1} | v_i, T\right]) \geq g(\mathsf{adv}(v_i, A_{r_{i+1}})) \tag{5.3}$$

Plugging (5.3) into (5.2) gives that

$$\begin{aligned}
\mathbb{E}\left[\mathrm{Var}(B_{i+1} | v_i)\right] &\geq \mathbb{E}_{v_i}\left[\mathrm{Var}_T(\mathbb{E}\left[B_{i+1} | v_i, T\right])\right] \geq \mathbb{E}_{v_i}\left[g(\mathsf{adv}(v_i, A_{r_{i+1}}))\right] \\
&\geq g\left(\mathbb{E}_{v_i}\left[\mathsf{adv}(v_i, A_{r_{i+1}})\right]\right) = g(\mathbb{E}[A_{i+1}]) \qquad (5.4) \\
&\geq g\left(\frac{\varepsilon}{2}\right) = \frac{\varepsilon^3}{1152} + o(\varepsilon^3) \qquad (5.5)
\end{aligned}$$

where (5.4) follows from the Jensen inequality and (5.5) follows from Lemma 5.3.2 and the fact that g is an increasing function. Note that the right hand side is at least $\frac{\varepsilon}{1200}$ for sufficiently small ε. This completes the proof of Lemma 5.3.3 (With the exception of Lemma 5.3.4). □

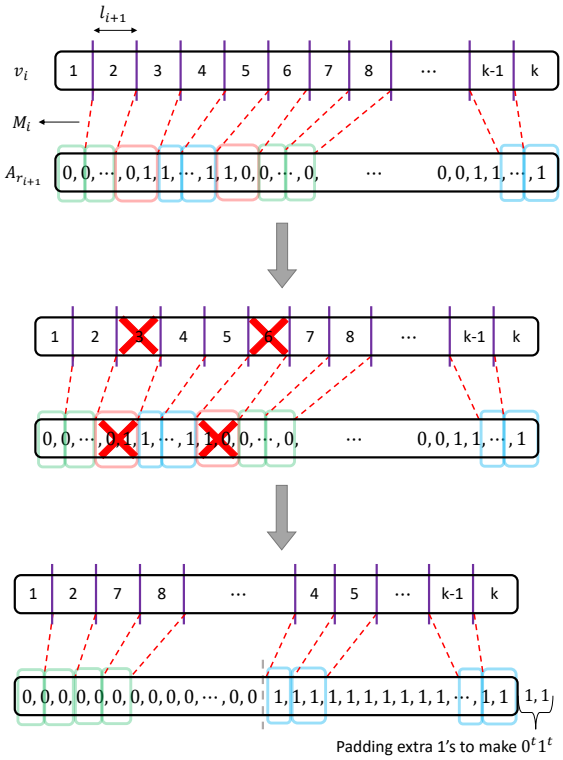


Figure 5.3: Three steps of transformation in Lemma 5.3.4.

With Lemma 5.3.3 proved, one can easily prove Theorem 5.1.4.

Proof of Theorem 5.1.4. Since $\mathrm{Var}(B_{i+1}) \geq \mathrm{Var}(B_i) + \varepsilon^3/1200$, we have that

$$\mathrm{Var}(B_k) \geq \mathrm{Var}(B_1) + (k-1)\frac{\varepsilon^3}{1200} \geq \frac{(k-1)\varepsilon^3}{1200}.$$

If $k > \frac{1200}{\varepsilon^3}$, the above inequality implies that $\mathrm{Var}(B_k) > 1$ which is impossible since B_k takes value in $[-1, 1]$. This contradiction implies that the list size $k \leq \frac{1200}{\varepsilon^3}$. □

We now proceed to the proof of Lemma 5.3.4.

5.3.1 Proof of Lemma 5.3.4

Consider v_i and the matching that yields the optimal advantage from v_i to $A_{r_{i+1}}$, denoted by M_i. We denote the substring of $A_{r_{i+1}}$ that is identified by the projection of v_i under M_i as $p = v_i \to M_i$. To simplify the analysis, we perform a series of transformations on v_i, M_i, and p that does not decrease adv_{M_i} except by a small quantity. Fig. 5.3 depicts the steps of this transformation described below.

1. First, we delete all substrings of U_e—i.e., substrings of length l_i in v_i whose projection contain both zeros and ones—from v_i.

2. We reorder the substrings of length l_{i+1} in v_i by shifting all U_0 substrings together and all U_1 substrings together. We accordingly shift the projections of these strings in p to the similar order. This was, the remainder of M_i from step 1 will be preserved as a valid matching between reordered strings.

3. At this point, string p consists of a stretch of zeros followed by a stretch of ones. If the length of two stretches are not equal, we add adequate zeros or ones to the smaller stretch to make p have the form of 0^t1^t.

To track the changes in adv_{M_i} during this transformation, we track how $|M_i|$, $|v_i|$ and $|p|$ change throughout the three steps mentioned above.

In the first step, a total of up to $|U_e|l_{i+1}$ elements are removed from v_i and M_i. Note that since the run length of $A_{r_{i+1}}$ is r_{i+1}, there can only be $\frac{|p|}{r_{i+1}}$ substrings in U_e. Therefore,

$$|U_e|l_{i+1} \leq \frac{|p|l_{i+1}}{r_{i+1}} = |p|\varepsilon^2 \leq 2\varepsilon^2|v_i|.$$

The second step preserves $|M_i|$, $|v_i|$ and $|p|$.

Finally, since p is a substring of $A_{r_{i+1}}$, the third step increases $|p|$ only by up to r_{i+1}. Note the run length of the $A_{r_{i+1}}$s and consequently l_{i+1}s are different by a multiplicative factor of at least $\frac{1}{\varepsilon^4}$ by the definition of the code $\mathcal{C}$. Therefore, $r_{i+1} = \frac{l_{i+1}}{\varepsilon^2} = \frac{l_{i+1}|v_i|}{\varepsilon^2|v_i|} = \frac{l_{i+1}|v_i|}{\varepsilon^2 l_i} \leq \varepsilon^2|v_i|$.

Overall, the value of the $\mathsf{adv}_{M_i} = \frac{3|M|-|p|-|v_i|}{|v_i|}$ can be affected by a maximum of $(3-1)\times 2\varepsilon^2|v_i| + \varepsilon^2|v_i| = 5\varepsilon^2|v_i|$ decrease in the numerator and $\varepsilon^2|v_i|$ decrease in the denominator. Therefore, the eventual advantage does not drop below $\mathsf{adv}_{M_i} - 5\varepsilon^2$. Let us denote the transformed versions of v_i, p, and M_i by $\bar{v}_i$, $\bar{p}$, and $\bar{M}_i$ respectively. We have shown that

$$\mathsf{adv}_{\bar{M}_i} \geq \mathsf{adv}_{M_i} - 5\varepsilon^2. \tag{5.6}$$

Further, let $\bar{v}_i = (\bar{v}_i^0, \bar{v}_i^1)$ so that $\bar{v}_i^0$ and $\bar{v}_i^1$ respectively correspond to the part of $\bar{v}_i$ that is mapped to 0^t and 1^t under $\bar{M}_i$. Consider the matching between $\bar{v}_i$ and $\bar{p}$ that connects as many zeros as possible between the $\bar{v}_i^0$ and 0^t and as many ones as possible between the $\bar{v}_i^1$ to 1^t portion of $\bar{p}$. Clearly, the size of $\bar{M}_i$ cannot exceed the size of this matching and therefore,

$$\mathsf{adv}_{\bar{M}_i} \leq \frac{3\left[\min\{t, \mathsf{count}_0(\bar{v}_i^0)\} + \min\{t, \mathsf{count}_1(\bar{v}_i^1)\}\right] - |\bar{v}_i| - 2t}{|\bar{v}_i|} \tag{5.7}$$

Note that as long as $t < \mathsf{count}_0(\bar{v}_i^0)$ or $t < \mathsf{count}_1(\bar{v}_i^1)$, increasing t in the right hand side term does not make it smaller. Therefore, the inequality (5.7) holds for $t = \max_{j\in\{0,1\}}\{\mathsf{count}_j(\bar{v}_i^j)\}$. Without loss of generality, assume that $\mathsf{count}_0(\bar{v}_i^0) \leq \mathsf{count}_1(\bar{v}_i^1)$

and set $t = \mathsf{count}_1(\bar{v}_i^1)$. Then we have the following.

$$\begin{aligned}
& \mathsf{adv}_{\bar{M}_i} \leq \frac{3\mathsf{count}_0(\bar{v}_i^0) + \mathsf{count}_1(\bar{v}_i^1) - |\bar{v}_i|}{|\bar{v}_i|} \\
\Rightarrow \quad & \mathsf{adv}_{\bar{M}_i} \leq \frac{3\frac{1-\mathsf{bias}(\bar{v}_i^0)}{2}|\bar{v}_i^0| + \frac{1+\mathsf{bias}(\bar{v}_i^1)}{2}|\bar{v}_i^1| - (|\bar{v}_i^0| + |\bar{v}_i^1|)}{|\bar{v}_i|} \\
\Rightarrow \quad & 2\mathsf{adv}_{\bar{M}_i}|\bar{v}_i| \leq 3(1 - \mathsf{bias}(\bar{v}_i^0))|\bar{v}_i^0| + (1 + \mathsf{bias}(\bar{v}_i^1))|\bar{v}_i^1| - 2(|\bar{v}_i^0| + |\bar{v}_i^1|) \\
\Rightarrow \quad & 2\mathsf{adv}_{\bar{M}_i}|\bar{v}_i| \leq \left[1 - 3\mathsf{bias}(\bar{v}_i^0)\right]|\bar{v}_i^0| - \left[1 - \mathsf{bias}(\bar{v}_i^1)\right]|\bar{v}_i^1| \qquad (5.8)
\end{aligned}$$

We claim that the above inequality leads to the fact that $|\mathsf{bias}(\bar{v}_i^1) - \mathsf{bias}(\bar{v}_i^0)| \geq \mathsf{adv}_{\bar{M}_i}/3$. Assume for contradiction that this is not the case. Therefore, replacing the term $\mathsf{bias}(\bar{v}_i^0)$ with $\mathsf{bias}(\bar{v}_i^1)$ in (5.8) does not change the value of the right hand side by any more than $|\bar{v}_i| \cdot \mathsf{adv}_{\bar{M}_i}$. Same holds true with replacing the term $\mathsf{bias}(\bar{v}_i^1)$ with $\mathsf{bias}(\bar{v}_i^0)$ in (5.8). This implies that, with $b^* = \max\{\mathsf{bias}(\bar{v}_i^0), \mathsf{bias}(\bar{v}_i^1)\}$, we have that

$$\begin{aligned}
& \mathsf{adv}_{\bar{M}_i}|\bar{v}_i| \leq (1 - 3b^*) \cdot |\bar{v}_i^0| - (1 - b^*)|\bar{v}_i^1| \\
\Rightarrow \quad & (1 - b^*)|\bar{v}_i^1| < (1 - 3b^*)|\bar{v}_i^0| \qquad (5.9)
\end{aligned}$$

On the other hand, we assumed earlier (without loss of generality) that $\mathsf{count}_0(\bar{v}_i^0) \leq \mathsf{count}_1(\bar{v}_i^1)$. Therefore,

$$\begin{aligned}
& \mathsf{count}_0(\bar{v}_i^0) \leq \mathsf{count}_1(\bar{v}_i^1) \\
\Rightarrow \quad & \left(1 - \mathsf{bias}(\bar{v}_i^0)\right)|\bar{v}_i^0| \leq \left(1 + \mathsf{bias}(\bar{v}_i^1)\right)|\bar{v}_i^1| \\
\Rightarrow \quad & (1 - b^*)|\bar{v}_i^0| \leq (1 + b^*)|\bar{v}_i^1| \qquad (5.10)
\end{aligned}$$

Note that since $|b^*| \leq 1$, $(1 - b^*)^2 > (1 + b^*)(1 - 3b^*) \Rightarrow \frac{1-3b^*}{1-b^*} < \frac{1-b^*}{1+b^*}$. Multiplying the two sides of this inequality to the sides of (5.10) gives that

$$(1 - 3b^*)|\bar{v}_i^0| \leq (1 + b^*)|\bar{v}_i^1|$$

which contradicts (5.9). Therefore, we must have

$$|\mathsf{bias}(\bar{v}_i^1) - \mathsf{bias}(\bar{v}_i^0)| \geq \mathsf{adv}_{\bar{M}_i}/3.$$

Note that $\mathsf{bias}(\bar{v}_i^j) = \mathbb{E}\left[B_{i+1}|v_i, T = j\right]$ since $\mathsf{bias}(\bar{v}_i^j)$ is the average bias of all strings in U_j. Therefore, combining with (5.6), we have that

$$\left|\mathbb{E}\left[B_{i+1}|v_i, T = 0\right] - \mathbb{E}\left[B_{i+1}|v_i, T = 1\right]\right| \geq \frac{\mathsf{adv}(v_i, A_{r_{i+1}}) - 5\varepsilon^2}{3}. \qquad \square$$

5.4 Proof of Theorem 5.1.2: Concatenated InsDel Codes

We recall that the concatenation of an inner insdel code $\mathcal{C}_{\text{in}}$ over an alphabet of size $|\Sigma_{\text{in}}|$ and an outer insdel code, $\mathcal{C}_{\text{out}}$, over an alphabet of size $|\Sigma_{\text{out}}| = |\mathcal{C}_{\text{in}}|$ as a code over alphabet

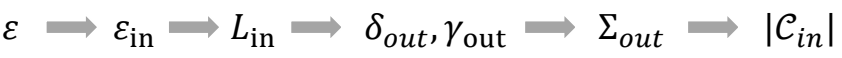


Figure 5.4: The order of determining parameters in the proof of Theorem 5.1.2.

Σ_{in}, is obtained by taking each codeword $x \in \mathcal{C}_{\text{out}}$, encoding each symbol of x with $\mathcal{C}_{\text{in}}$, and appending the encoded strings together to obtain each codeword of the concatenated code.

In this section, we will show that, concatenating an inner code $\mathcal{C}_{\text{in}}$ from Theorem 5.1.4 that can L_{in}-list decode from any γ fraction of insertions and δ fraction deletions when $2\delta+\gamma < 1-\varepsilon_{\text{in}}$ along with an appropriately chosen outer code $\mathcal{C}_{\text{out}}$ from Theorem 4.1.1, one can obtain an infinite family of constant-rate insertion-deletion codes that are efficiently list-decodable from any γ fraction of insertions and δ fraction of deletions as long as $2\delta+\gamma < 1-\varepsilon$ for $\varepsilon = \frac{16}{5}\varepsilon_{\text{in}}$.

5.4.1 Construction of the Concatenated Code

We start by fixing some notation. Let $\mathcal{C}_{\text{out}}$ be able to L_{out}-list decode from δ_{out} fraction of deletions and γ_{out} fraction of insertions. Further, let us indicate the block sizes of $\mathcal{C}_{\text{out}}$ and $\mathcal{C}_{\text{in}}$ with n_{out} and $n_{\text{in}} = \lceil \log |\Sigma_{\text{out}}| \rceil$.

To construct our concatenated codes, we utilize Theorem 4.1.1 to obtain an efficient family of codes $\mathcal{C}_{\text{out}}$ over alphabet Σ_{out} of size $O_{\gamma_{\text{out}},\delta_{\text{out}}}(1)$ that is L_{out}-list decodable from any δ_{out} fraction of deletions and γ_{out} fraction of insertions for appropriate parameters δ_{out} and γ_{out} that we determine later. We then concatenate any code in $\mathcal{C}_{\text{out}}$ with an instance of the binary list-decodable codes from Theorem 5.1.4, C_{in}, with parameter $n_{\text{in}} = \lceil \log |\Sigma_{\text{out}}| \rceil$ and a properly chosen ε_{in}. We will determine appropriate values for all these parameters given ε when describing the decoding procedure in Section 5.4.2. Fig. 5.4 shows the order of determining all parameters. We remark that the following two properties for the utilized inner and outer codes are critical to this order of fixing parameters:

1. The alphabet size of the family of codes used as the outer code only depends on δ_{out} and γ_{out} and is independent of the outer block size n_{out}. (See Theorem 4.1.1)

2. The list size of the family of codes used as the inner code, L_{in}, merely depends on parameter ε_{in} in Theorem 5.1.4 and is independent of the size of the code or its block length, i.e., $|\mathcal{C}_{\text{in}}|$ or n_{in}.

5.4.2 Decoding Procedure and Determining Parameters

We now analyze the resulting family of codes and choose the undetermined parameters along the way of describing the decoding procedure. A pseudo-code of the decoding procedure is available in Algorithm 5. Let $\mathcal{C}$ be a binary code with block length n that is obtained from the above-mentioned concatenation. Take the codeword $x \in C$ and split it into *blocks* of length n_{in}. Note that each such block corresponds to the encoding of some

symbol in Σ_{out} under $\mathcal{C}_{\text{in}}$. Let x' be a string obtained by applying $n\gamma$ insertions and $n\delta$ deletions into x where $n = n_{\text{in}}n_{\text{out}}$ and $\gamma + 2\delta < 1 - \varepsilon$. For each block of x, we define the error count to be the total number of insertions that have occurred in that block plus twice the number of deleted symbols in it. Clearly, the average value of error count among all blocks is $n_{\text{in}}(\gamma+2\delta) < n_{\text{in}}(1-\varepsilon)$. By a simple averaging, at least $\left(1 - \frac{1-\varepsilon}{1-\varepsilon/4}\right) n_{\text{out}} \geq \frac{3\varepsilon}{4} \cdot n_{\text{out}}$ of those blocks have an error count of $n_{\text{in}}(1 - \frac{\varepsilon}{4})$ or less. Let us call the set of all such blocks S.

Further, we partition S into smaller sets based on the number of deletions occurring in the blocks of S. Let $S_i \in S$ be the subset of blocks in S for which the number of deletions is in $\left[n_{\text{in}} \cdot \frac{\varepsilon}{16} \cdot (i-1), n_{\text{in}} \cdot \frac{\varepsilon}{16} \cdot i\right)$ for $i = 1, 2, \cdots, 8/\varepsilon$[1]. The following two properties hold true:

1. All blocks in S_i suffer from at least $n_{\text{in}} \cdot \frac{\varepsilon}{16} \cdot (i-1)$ deletions. Further, they can suffer from up to $n_{\text{in}} \cdot \left(1 - \frac{\varepsilon}{4} - \frac{2\varepsilon}{16} \cdot (i-1)\right)$ insertions. Therefore, they all appear as substrings of length $n_{\text{in}} \cdot \left(2 - \frac{\varepsilon}{4} - \frac{3\varepsilon}{16} \cdot (i-1)\right)$ or less in x'.

2. We have that $S = \dot{\bigcup}_{i=1}^{8/\varepsilon} S_i$. By the Pigeonhole principle, for some $i^* \in [1, 8/\varepsilon]$, $|S_{i^*}| \geq \frac{3\varepsilon^2}{32} n_{\text{out}}$.

Our decoding algorithm consists of $8/\varepsilon$ rounds each consisting of two phases of inner and outer decoding. During the first phase of each round $i = 1, 2, \cdots, 8/\varepsilon$, the algorithm uses the decoder of the inner code on x' to construct a string T_i over alphabet Σ_{out} and then, in the second phase, uses the decoder of the outer code on input T_i to obtain a list $List_i$ of size L_{out}. In the end, the decoding algorithm outputs the union of all such lists $\bigcup_i List_i$.

Algorithm 5 Decoder of the Concatenated Code

procedure CONCATENATED-DECODER($x', \varepsilon, n_{\text{in}}, n_{\text{out}}, \text{Dec}_{\mathcal{C}_{\text{in}}}(\cdot), \text{Dec}_{\mathcal{C}_{\text{out}}}(\cdot)$)
  Output $\leftarrow \emptyset$
  for $i \in \left\{1, 2, \cdots, \frac{8}{\varepsilon}\right\}$ **do** ▷ Round i
    $w \leftarrow \left\lfloor \frac{n_{\text{in}}(2-\varepsilon/4-3\varepsilon(i-1)/16)}{n_{\text{in}}\varepsilon/16} \right\rfloor + 1$ ▷ Length of the sliding window is $w \cdot \frac{n_{\text{in}}\varepsilon}{16}$.
    $T_i \leftarrow$ empty string
    for $j \in \left\{1, 2, \cdots, \frac{|x'|}{n_{\text{in}}\varepsilon/16} - w\right\}$ **do** ▷ Phase I: Inner Decoding
      $List \leftarrow \text{Dec}_{\mathcal{C}_{\text{in}}}\left(x'\left[\frac{n_{\text{in}}\varepsilon}{16} \cdot j, \frac{n_{\text{in}}\varepsilon}{16} \cdot (j+w)\right]\right)$
      Pad symbols of Σ_{out} corresponding to the elements of $List$ to the right of T_i.
    Output $\leftarrow$ Output $\cup$ $\text{Dec}_{\mathcal{C}_{\text{out}}}(T_i)$ ▷ Phase II: Outer Decoding
  return Output

[1]Note that the fraction of deletions cannot exceed $\frac{1}{2}$ assuming $n_{\text{in}}(\gamma + 2\delta) < n_{\text{in}}(1 - \varepsilon)$.

Description of Phase I (Inner Decoding) We now proceed to the description of the first phase in each round $i \in \{1, 2, \cdots, 8/\varepsilon\}$. In the construction of T_i, we aim for correctly decoding the blocks in S_i. As mentioned above, all such blocks appear in x' in a substring of length $n_{\text{in}} \cdot \left(2 - \frac{\varepsilon}{4} - \frac{3\varepsilon}{16} \cdot (i-1)\right)$ or less.

Having this observation, we run the deocoder of the inner code on substrings of x' of form $x'\left[\frac{n_{\text{in}}\varepsilon}{16} \cdot j, \frac{n_{\text{in}}\varepsilon}{16} \cdot (j+w)\right]$ for all $j = 1, 2, \cdots, \frac{|x'|}{n_{\text{in}}\varepsilon/16} - w$ where

$$w = \left\lfloor \frac{n_{\text{in}}(2 - \varepsilon/4 - 3\varepsilon(i-1)/16)}{n_{\text{in}}\varepsilon/16} \right\rfloor + 1.$$

One can think of such substrings as a *window* of size $w \cdot \frac{n_{\text{in}}\varepsilon}{16}$ that slides in $\frac{n_{\text{in}}\varepsilon}{16}$ increments.

Note that each block B in S_i appears within such window and is far from it by, say, D_B deletions and no more than $n_{\text{in}}\left(1 - \frac{\varepsilon}{4}\right) - 2D_B + \frac{n_{\text{in}}\varepsilon}{16}$ insertions where the additional $\frac{n_{\text{in}}\varepsilon}{16}$ term in insertion count comes from the extra symbols around the block in the fixed sized window. As long as the fraction of insertions plus twice the fraction of deletions that are needed to convert a block of S_i into its corresponding window does not exceed $1 - \varepsilon_{\text{in}}$, the output of the inner code's decoder for input $x'\left[\frac{n_{\text{in}}\varepsilon}{16} \cdot j, \frac{n_{\text{in}}\varepsilon}{16} \cdot (j+w)\right]$ will contain the block B of S_i. So, we choose ε_{in} such that

$$\begin{aligned} & n_{\text{in}}\left(1 - \frac{\varepsilon}{4}\right) - 2D_B + \frac{n_{\text{in}}\varepsilon}{16} + 2D_B \leq n_{\text{in}}(1 - \varepsilon_{\text{in}}) \qquad (5.11) \\ \Leftrightarrow \quad & n_{\text{in}}(1 - 3\varepsilon/16) \leq n_{\text{in}}(1 - \varepsilon_{\text{in}}) \\ \Leftrightarrow \quad & \varepsilon_{\text{in}} \leq \frac{3}{16}\varepsilon \end{aligned}$$

Now, each element in the output list corresponds to some codeword of the inner code and, therefore, some symbol in Σ_{out}. For each run of the decoder of the inner code, we take the corresponding symbols of Σ_{out} and write them back-to-back in arbitrary order. Then, we append all such strings in the increasing order of j to obtain T_i.

Description of Phase II (Outer Decoding) Note that the length of T_i is at most $\frac{|x'|}{n_{\text{in}}\varepsilon/16} L_{\text{in}} \leq \frac{2n_{\text{in}}n_{\text{out}}}{n_{\text{in}}\varepsilon/16} L_{\text{in}} = n_{\text{out}} \cdot \frac{32}{\varepsilon} L_{\text{in}}$. Further, T_i contains symbols corresponding to all blocks of S_i as a subsequence (i.e., in the order of appearance) except possibly the ones that appear in the same run of the inner decoder together. Since the fraction of deletions happening to each block in S_i is less than $\frac{1}{2}$ and the size of the inner decoding sliding window is no more than $2n_{\text{in}}$, the number of blocks of S_i that can appear in the same window in the first phase is at most 4. This gives that T_i has a common subsequence of size at least $\frac{|S_i|}{4}$ with the codeword of the outer code.

We mentioned earlier that for some i^*, $|S_{i^*}| \geq \frac{3\varepsilon^2}{32} n_{\text{out}}$. Therefore, for such i^*, T_{i^*} is different from x by up to a $1 - \frac{3\varepsilon^2}{128}$ fraction of deletions and $\frac{32}{\varepsilon} L_{\text{in}}$ fraction of insertions. Therefore, by taking $\delta_{\text{out}} = 1 - \frac{3\varepsilon^2}{128}$, $\gamma_{\text{out}} = \frac{32}{\varepsilon} L_{\text{in}} = O\left(\frac{1}{\varepsilon^4}\right)$, and using each T_i as an input to the decoder of the outer code in the second phase, x will certainly appear in the outer output list for some T_i. (Specifically, for $i = i^*$.)

5.4.3 Remaining Parameters

As shown in Section 5.4.2, we need a list-decodable code as outer code that can list-decode from $\delta_{\text{out}} = 1 - \frac{3\varepsilon^2}{128}$ fraction of deletions and $\gamma_{\text{out}} = \frac{32}{\varepsilon} L_{\text{in}} = O\left(\frac{1}{\varepsilon^4}\right)$ fraction of insertions. To obtain such codes we use Theorem 4.1.1 with parameters $\gamma = \frac{32}{\varepsilon} L_{\text{in}}$ and $\epsilon = \frac{3\varepsilon^2}{256}$. This implies that the rate of the outer code is $r_{\text{out}} = \frac{3\varepsilon^2}{256} = O(\varepsilon^2)$, it is $L_{\text{out}} = O_\varepsilon(\exp(\exp(\exp(\log^* n))))$ list-decodable, and can be defined over an alphabet size of $|\Sigma_{\text{out}}| = e^{O\left(\frac{1}{\varepsilon^{10}} \log \frac{1}{\varepsilon^8}\right)}$.

Consequently, $|\mathcal{C}_{in}| = \log |\Sigma_{\text{out}}| = O\left(\frac{1}{\varepsilon^{10}} \log \frac{1}{\varepsilon}\right)$. Note that in Theorem 5.1.4, the block length of the inner code can be chosen independently of its list size as the list size only depends on ε_{in}. This is a crucial quality in our construction since in our analysis ε_{in} and L_{in} are fixed first and then $|C_{\text{in}}|$ is chosen depending on the properties of the outer code.

As the decoder of the outer code is used $\frac{8}{\varepsilon}$ times in the decoding of the concatenated code, the list size of the concatenated code will be $L = \frac{8}{\varepsilon} \cdot L_{\text{out}} = O_\varepsilon(\exp(\exp(\exp(\log^* n))))$. The rate of the concatenated code is

$$r = r_{\text{out}} r_{\text{in}} = O\left(\varepsilon^2 \cdot \frac{\log \log |\mathcal{C}_{\text{in}}|}{n_{\text{in}}}\right) = O\left(\varepsilon^2 \cdot \frac{\log \log |\mathcal{C}_{\text{in}}|}{(1/\varepsilon^4)^{|\mathcal{C}_{\text{in}}|}}\right) = e^{-O\left(\frac{1}{\varepsilon^{10}} \log^2 \frac{1}{\varepsilon}\right)}.$$

Finally, since the outer code is efficient and the inner code is explicit and can be decoded by brute-force in $O_\varepsilon(1)$ time, the encoding and decoding procedures run in polynomial time. This concludes the proof of Theorem 5.1.2.

5.5 Extension to Larger Alphabets

In this section we extend the results presented so far to q-ary alphabets where $q > 2$.

5.5.1 Feasibility Region: Upper Bound

For an alphabet of size q, no positive-rate family of deletion codes can protect against $1 - \frac{1}{q}$ fraction of errors since, with that many deletions, an adversary can simply delete all but the most frequent symbol of any codeword. Similarly, for insertion codes, it is not possible to achieve resilience against $q - 1$ fraction of errors as adversary would be able to turn any codeword $x \in q^n$ to $(1, 2, \cdots, q)^n$.

The findings of the previous sections on binary alphabets might suggest that the feasibility region for list-decoding is the region mapped out by these two points, i.e., $\frac{\delta}{1-\frac{1}{q}} + \frac{\gamma}{q-1} < 1$. However, this conjecture turns out to be false. The following theorem provides a family of counterexamples.

Theorem 5.5.1. *For any alphabet size q and any $i = 1, 2, \cdots, q$, no positive-rate q-ary infinite family of insertion-deletion codes can list-decode from $\delta = \frac{q-i}{q}$ fraction of deletions and $\gamma = \frac{i(i-1)}{q}$ fraction of insertions.*

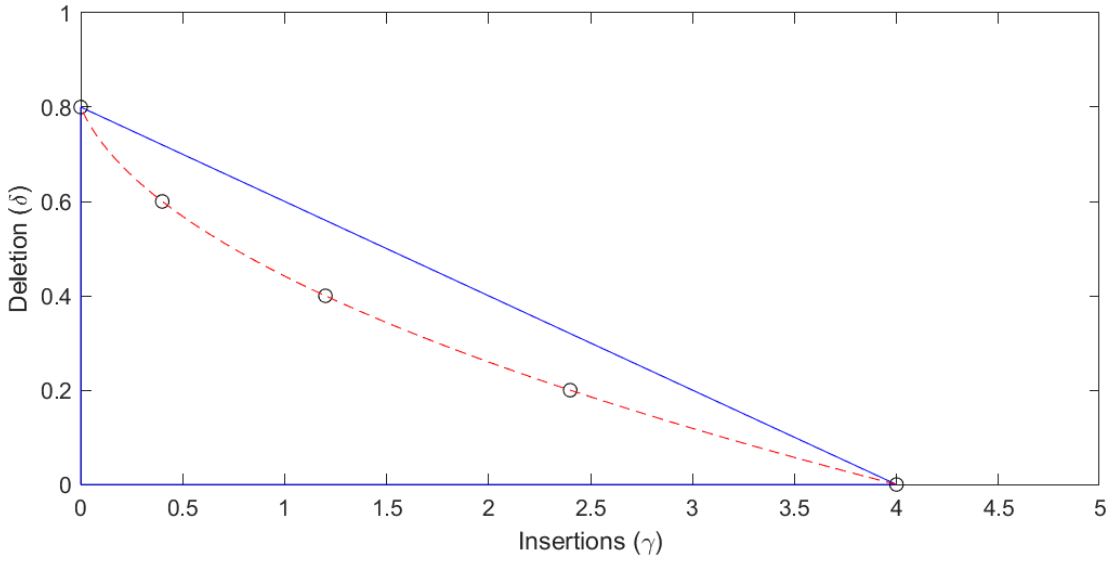


Figure 5.5: Infeasible points inside the conjectured feasibility region. (Illustrated for $q = 5$)

Proof. Take a codeword $x \in [q]^n$. With $\delta n = \frac{q-i}{q} \cdot n$ deletions, the adversary can delete the $q - i$ least frequent symbols to turn x into $x' \in \Sigma_d^{n(1-\delta)}$ for some $\Sigma_d = \{\sigma_1, \cdots, \sigma_i\} \subseteq [q]$. Then, with $\gamma n = n(1-\delta)(i-1) = n\frac{i(i-1)}{q}$ insertions, it can turn x' into $[\sigma_1, \sigma_2, \cdots, \sigma_i]^{n(1-\delta)}$. Such adversary only allows $O(1)$ amount of information to pass to the receiver. Hence, no such family of codes can yield a positive rate. □

Note that all points $(\gamma, \delta) = \left(\frac{i(i-1)}{q}, \frac{q-i}{q}\right)$ are located on a second degree curve inside the conjectured feasibility region $\frac{\delta}{1-\frac{1}{q}} + \frac{\gamma}{q-1} < 1$ (see Fig. 5.5). Our next step is to show that the actual feasibility region is a subset of the polygon outlined by these points.

Theorem 5.5.2. *For any positive integer $q > 2$, define F_q as the concave polygon defined over vertices $\left(\frac{i(i-1)}{q}, \frac{q-i}{q}\right)$ for $i = 1, \cdots, q$ and $(0, 0)$. (see Fig. 5.1). F_q does not include the border except the two segments $[(0, 0), (q - 1, 0))$ and $\left[(0, 0), \left(0, 1 - \frac{1}{q}\right)\right)$. Then, for any pair of positive real numbers $(\gamma, \delta) \notin F_q$, there exists no infinite family of q-ary codes with positive rate that can correct from δ fraction of deletions and γ fraction of insertions.*

Proof. In order to prove this, it suffices to show that for any pair of consecutive vertices on the polygon like $p_i = \left(\frac{i(i-1)}{q}, \frac{q-i}{q}\right)$ and $p_{i+1} = \left(\frac{i(i+1)}{q}, \frac{q-i-1}{q}\right)$, the entirety of the segment between p_i and p_{i+1} lie outside of the feasibility region. To this end, we show that for any $i = 1, 2, \cdots, q - 1$ and $\alpha \in (0, 1)$, no family of codes with positive rate is list-decodable from $(\gamma_0, \delta_0) = \alpha p_i + (1 - \alpha)p_{i+1}$ fraction of insertions and deletions. Note that in Theorem 5.5.1 we proved the infeasibility of the vertices of F_q by providing a strategy for the adversary to convert any string into one out of a set of size $O_q(1)$ using the corresponding amount of insertions and deletions. To finish the proof, we similarly present a strategy for

the adversary that is obtained by a simple time sharing between the ones used to show infeasibility at p_i and p_{i+1} in Theorem 5.5.1.

Consider a codeword $x \in [q]^n$. As shown in Theorem 5.5.1, the adversary can utilize $n\alpha \cdot p_i$ errors to convert the first αn symbols of x into a string of form $[\sigma_1, \sigma_2, \cdots, \sigma_i]^{n\alpha \cdot \frac{i}{q}}$ where $\{\sigma_1, \sigma_2, \cdots, \sigma_i\} \subseteq \Sigma$. Similarly, the remaining $n(1-\alpha)p_{i+1}$ errors can be utilized to turn the last $(1-\alpha)n$ symbols of x into a string of the form $[\sigma'_1, \sigma'_2, \cdots, \sigma'_{i+1}]^{n(1-\alpha)\cdot\frac{i+1}{q}}$ where $\{\sigma_1, \sigma_2, \cdots, \sigma_{i+1}\} \subseteq \Sigma$. Note that there are no more than $\binom{q}{i} i! \cdot \binom{q}{i+1}(i+1)! = O_q(1)$ of such strings. Therefore, for any given positive rate code, there exists one string of the above-mentioned form which is (γ_0, δ_0)-close to exponentially many codewords and, thus, no positive-rate family of codes is list-decodable from (γ_0, δ_0) fraction of insertions and deletions. □

5.5.2 Feasibility Region: Exact Characterization

Finally, we will show that the feasibility region is indeed equal to the region F_q described in Theorem 5.5.2. The proof closely follows the steps taken for the binary case but is significantly more technical. We first formally define q-ary Bukh-Ma codes and show they are list-decodable as long as the error rate lies in F_q and then use the concatenation in Section 5.4 to obtain Theorem 5.1.3.

Theorem 5.5.3. *For any integer $q \geq 2$, $\varepsilon > 0$, and sufficiently large n, let $C^q_{n,\varepsilon}$ be the following Bukh-Ma code:*

$$C^q_{n,\varepsilon} = \left\{ (0^r 1^r \cdots q^r)^{\frac{n}{qr}} \,\middle|\, r = \left(\frac{1}{\varepsilon^4}\right)^k, k < \log_{1/\varepsilon^4} n \right\}.$$

For any $(\gamma, \delta) \in (1-\varepsilon)F_q$ it holds that $C^q_{n,\varepsilon}$ is list decodable from any δn deletions and γn insertions with a list size of $O\left(\frac{q^5}{\varepsilon^2}\right)$.

We remark that in the case of $q = 2$, Theorem 5.5.3 improves over Theorem 5.1.4 in terms of the dependence of the list size on ε.

Proof Sketch for Theorem 5.5.3

To prove Theorem 5.5.3, we show that Bukh-Ma codes are list-decodable as long as the error rate (γ, δ) lies beneath the line that connects a pair of consecutive non-zero vertices of F_q.

In other words, for any pair of points $\left(\frac{i(i-1)}{q}, \frac{q-i}{q}\right)$ and $\left(\frac{i(i+1)}{q}, \frac{q-i-1}{q}\right)$ we consider the line passing through them (see Fig. 5.6), i.e.,

$$\gamma + (2i)\delta = \frac{(2q-1)i - i^2}{q}, \quad i = 1, \cdots, q-1 \tag{5.12}$$

and show that as long as $\gamma + (2z)\delta \leq (1-\varepsilon)\frac{(2q-1)z - z^2}{q}$ for some $z \in \{1, \cdots, q-1\}$, Bukh-Ma

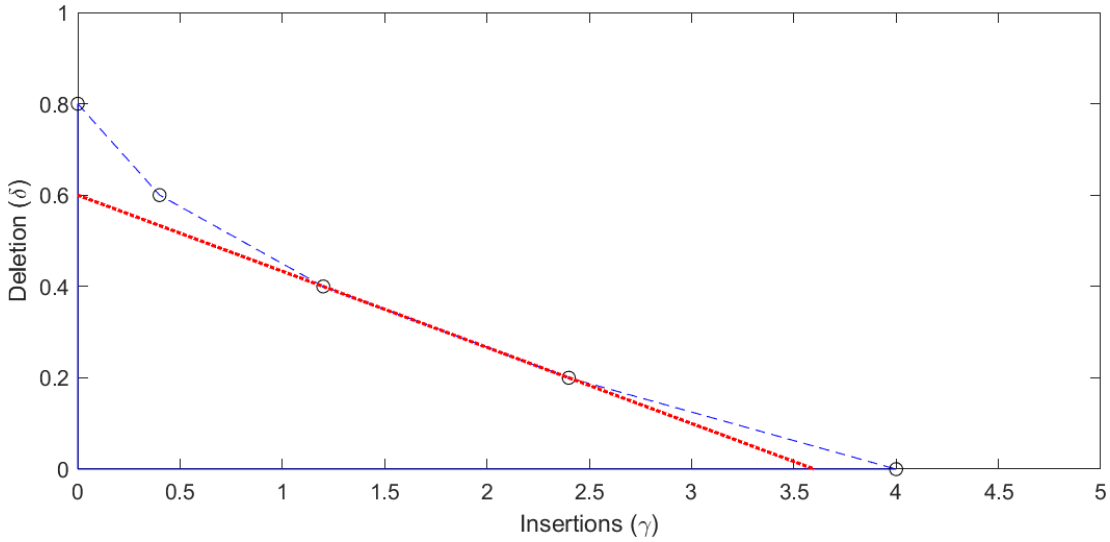


Figure 5.6: In the feasibility region for $q = 5$, the line passing through $(1.2, 0.4)$ and $(1.8, 0.3)$ (indicated with red dotted line) is characterized as $\gamma + 6\delta \leq 3.6$. (Corresponding to $i = 3$ in Eq. (5.12))

codes are list-decodable. Note that the union of such areas is equal to $(1 - \varepsilon)F_q$.

The analysis for each line follows the arguments for the binary case. Namely, we assume that k codewords can be converted to some center string v via (γ, δ) fraction of errors. Then, using an appropriate advantage notion and considering some coupled statistic processes obtained by sampling substrings, we show that k is bounded above by some $O_q\left(\mathsf{poly}(1/\varepsilon)\right)$.

The only major difference is that the notion of bias cannot be directly used for q-ary alphabets. In this general case, instead of keeping track of the variance of the bias, we keep track of the sum of the variances of the frequency of the occurrence of each symbol. We show that this quantity increases by some constant after each substring sampling (analogous to Lemma 5.3.3) by showing that a positive advantage requires that the frequency of occurrence of at least one of the symbols to be ε-different for two different values of the random variable T (analogous to Lemma 5.3.4). The rest of this section contains more formal description of generalized notions and proofs for generalized q-ary claims.

5.5.3 Generalized Notation and Preliminary Lemmas

To prove Theorem 5.5.3, we need to generalize some of the notions and respective preliminary lemmas for the binary case. We start with defining ith order advantage.

Definition 5.5.4 (ith order q-ary advantage of matching M)**.** *For a pair of positive integers $i < q$, a pair of q-ary strings a and b, and a matching M between a and b, we define* ith order q-ary advantage of a to b *as follows:*

$$\mathsf{adv}_M^{q,i}(a, b) = \frac{(2i+1)|M| - |a| - \frac{i+i^2}{q} \cdot |b|}{|a|}$$

Note that the notion of advantage utilized for the binary case is obtained for $q = 2$ and $i = 1$ in the above definition. The notions of ith order advantage between two strings (that is independent of a specific matching, i.e., $\mathsf{adv}^{q,i}(a, b)$) and infinite ith order advantage are defined in a similar manner to the binary case.

Remark 5.5.5. *In the same spirit as of the binary case, $\mathsf{adv}^{q,i}_M(a, b)$ is simply the value of*

$$|b| \left(\frac{(2q-1)i - i^2}{q} - (2i)\delta_M - \gamma_M \right)$$

normalized by the length of a. Indeed,

$$\begin{aligned}
\mathsf{adv}^{q,i}_M(a,b) &= \frac{(2i+1)|M| - |a| - \frac{i+i^2}{q} \cdot |b|}{|a|} \\
&= \frac{(2i+1)|b|(1-\delta_M) - |b|(1-\delta_M+\gamma_M) - \frac{i+i^2}{q} \cdot |b|}{|a|} \\
&= \frac{|b|}{|a|} \cdot \left[(2i+1)(1-\delta_M) - (1-\delta_M+\gamma_M) - \frac{i+i^2}{q} \right] \\
&= \frac{|b|}{|a|} \cdot \left(\frac{(2q-1)i - i^2}{q} - (2i)\delta_M - \gamma_M \right).
\end{aligned}$$

Lemma 5.5.6. *If for strings a and b, $\mathsf{adv}^{q,i}(a, b) \geq 0$, then $|a|$ and $|b|$ are within a q factor of each other.*

Proof. $\mathsf{adv}^{q,i}(a, b) \geq 0$ implies that for some matching M,

$$\begin{aligned}
& adv^{q,i}_M \geq 0 \Rightarrow |a| + \frac{i+i^2}{q} \cdot |b| \leq (2i+1)|M| \\
\Rightarrow \quad & q|a| + (i+i^2) \cdot |b| \leq q(2i+1)|M| \leq q(2i+1)\min(|a|,|b|) \qquad (5.13)
\end{aligned}$$

Now if $|a| \leq |b|$, (5.13) gives

$$q|a| + (i+i^2)|b| \leq q(2i+1)|a| \Rightarrow |b| \leq \frac{2q}{i+1} \cdot |a| \leq q|a|$$

and if $|b| < |a|$, (5.13) gives

$$q|a| + (i+i^2)|b| \leq q(2i+1)|b| \Rightarrow |a| \leq \frac{2iq + q - i - i^2}{q} \cdot |b| \leq q|b|.$$

□

Definition 5.5.7 (q-ary Alternating Strings)**.** *For any positive integer r, we define the infinite q-ary alternating string of run-length r as $A^q_r = (1^r 2^r \cdots q^r)^\infty$ and denote its prefix of length l by $A^q_{r,l} = A^q_r[1, l]$.*

5.5.4 Proof of Theorem 5.5.3

As mentioned before, Theorem 5.5.3 can be restated as follows.

Theorem 5.5.8 (Restatement of Theorem 5.5.3)**.** *For any integer $q \geq 2$, $\varepsilon > 0$, sufficiently large n, and any $z \in \{1, 2, \cdots, q-1\}$, the Bukh-Ma code $C^n_{n,\varepsilon}$ from Theorem 5.5.3 is list decodable from any δn deletions and γn insertions with a list size $O\left(q^5/\varepsilon^2\right)$ as long as $\gamma + (2z)\delta \leq (1-\varepsilon)\frac{(2q-1)z-z^2}{q}$.*

To prove this restated version, once again, we follow the steps taken for the proof of Theorem 5.1.4 and assume for the sake of contradiction that there exists a string v and $k = \Omega\left(\frac{q^5}{\varepsilon^2}\right)$ members of $C^q_{n,\varepsilon}$ like $A^q_{r_1,n}, A^q_{r_2,n}, \cdots, A^q_{r_k,n}$, so that each $A^q_{r_i,n}$ can be converted to v with I_i insertions and D_i deletions where $I_i + (2z)D_i \leq (1-\varepsilon)\frac{(2q-1)z-z^2}{q} \cdot n$. We define the indices in a way that $r_1 > r_2 > \cdots > r_k$. Given the definition of $C^q_{n,\varepsilon}$, $r_i \geq \frac{r_{i+1}}{\varepsilon^4}$.

Given Remark 5.5.5 and Lemma 5.5.6, an argument similar to the one presented in Lemma 5.3.1 shows that for all these codewords, $\mathsf{adv}^{q,z}(v, A^q_{r_i,n}) \geq \frac{\varepsilon}{q}$.

We define the following stochastic processes similar to the binary case. We split the string v into substrings of size $l_1 = r_1\varepsilon^2$, pick one uniformly at random and denote it by v_1. We define random variable $A_1 = \mathsf{adv}^{q,z}(v_1, A^q_{r_1})$ and random variables F^p_1 for $p = 1, 2, \cdots, q$ as the frequency of the occurrence of symbol p in v_1. In other words,

$$F^p_1 = \frac{\mathsf{count}_p(v_1)}{|v_1|}.$$

We continue this process for $j = 2, 3, \cdots, k$ by splitting each v_{j-1} into substrings of length $l_j = r_j\varepsilon^2$, picking v_j uniformly at random, and defining $A_j = \mathsf{adv}^{q,z}(v_j, A^q_{r_j})$ and $F^p_j = \frac{\mathsf{count}_p(v_j)}{|v_j|}$ for all $p \in \{1, 2, \cdots, q\}$. We then define the sequence of real numbers $f_1, f_2, \cdots, f_k$ as follows:

$$f_i = \sum_{p=1}^{q} \mathrm{Var}(F^p_i).$$

This series of real numbers will play the role of $\mathrm{Var}(B_i)$ in the binary case.

Lemma 5.5.9. *The following hold for $A_1, A_2, \cdots, A_k$ and $F^p_1, F^p_2, \cdots, F^p_k$ for all $p \in \{1, 2, \cdots, q\}$.*

1. $\mathbb{E}[F^p_i] = F^p_{i-1}$
2. $\mathbb{E}[A_i] \geq \frac{\varepsilon}{q}$

Proof. Since v_i is a substring of v_{i-1} chosen uniformly at random, the overall frequency of symbol p is equal to the average frequency of its occurrence in each substrings. The second item can be derived as in Lemma 5.3.2. □

The next lemma mimics Lemma 5.3.3 for the binary case.

Lemma 5.5.10. *For the sequence $f_1, f_2, \cdots, f_k$, we have that*

$$f_{i+1} \geq f_i + \Omega\left(\frac{\varepsilon^2}{q^4}\right).$$

Using Lemma 5.5.10, Theorem 5.5.8 can be simply proved as follows.

Proof of Theorem 5.5.8. Note that each f_i is the summation of the variance of q random variables that take values in $[0, 1]$. Therefore, their value cannot exceed q. Since $f_{i+1} \geq f_i + \Omega(\varepsilon^2/q^4)$, the total length of the series, k, may not exceed $O\left(\frac{q^5}{\varepsilon^2}\right)$. This implies that the list size is $O\left(\frac{q^5}{\varepsilon^2}\right)$. □

We now present the proof of Lemma 5.5.10.

Proof of Lemma 5.5.10. To relate f_i and f_{i+1}, we utilize the law of total variance as follows:

$$\begin{aligned} \mathrm{Var}(F_{i+1}^p) &= \mathrm{Var}\left(\mathbb{E}[F_{i+1}^p|v_i]\right) + \mathbb{E}\left[\mathrm{Var}(F_{i+1}^p|v_i)\right] \\ &= \mathrm{Var}\left(F_i^p\right) + \mathbb{E}\left[\mathrm{Var}(F_{i+1}^p|v_i)\right] \end{aligned} \tag{5.14}$$

Equation (5.14) comes from the fact that the average frequency of symbol p in substrings of length l_{i+1} of v_i is equal to the frequency of p in v_i. Having this, we see that it suffices to show that $\mathbb{E}\left[\mathrm{Var}(F_{i+1}^p|v_i)\right] \geq \Omega\left(\varepsilon^2/q^4\right)$. Similar to Lemma 5.3.3 we define E_j for $j = 1, 2, \cdots, q$ and E_e respectively as the event that the projection of v_{i+1} falls inside a $j^{r_{i+1}}$ in $A_{r_{i+1}}$ or a string containing multiple symbols. We also define the random variable T out of $\{e, 1, 2, \cdots, q\}$ that indicates which one of these events is realized. Once again, we use the law of total variance to bound $\mathbb{E}\left[\mathrm{Var}(F_{i+1}^p|v_i)\right]$.

$$\begin{aligned} \mathbb{E}\left[\mathrm{Var}(F_{i+1}^p|v_i)\right] &= \mathbb{E}_{v_i}\Bigg[\mathrm{Var}_T(\mathbb{E}\left[F_{i+1}^p|v_i, T\right]) + \mathbb{E}_T\left[\mathrm{Var}(F_{i+1}^p|v_i, T)\right]\Bigg] \\ &\geq \mathbb{E}_{v_i}\left[\mathrm{Var}_T(\mathbb{E}\left[F_{i+1}^p|v_i, T\right])\right] \end{aligned} \tag{5.15}$$

Combining (5.14) and (5.15) gives

$$\begin{aligned} & \mathrm{Var}(F_{i+1}^p) \geq \mathrm{Var}\left(F_i^p\right) + \mathbb{E}_{v_i}\left[\mathrm{Var}_T\left(\mathbb{E}\left[F_{i+1}^p|v_i, T\right]\right)\right] \\ \Rightarrow \quad & \sum_{p=1}^{q} \mathrm{Var}(F_{i+1}^p) \geq \sum_{p=1}^{q} \mathrm{Var}\left(F_i^p\right) + \sum_{p=1}^{q} \mathbb{E}_{v_i}\left[\mathrm{Var}_T\left(\mathbb{E}\left[F_{i+1}^p|v_i, T\right]\right)\right] \\ \Rightarrow \quad & f_{i+1} \geq f_i + \sum_{p=1}^{q} \mathbb{E}_{v_i}\left[\mathrm{Var}_T\left(\mathbb{E}\left[F_{i+1}^p|v_i, T\right]\right)\right] \\ \Rightarrow \quad & f_{i+1} \geq f_i + \mathbb{E}_{v_i}\left[\sum_{p=1}^{q} \mathrm{Var}_T\left(\mathbb{E}\left[F_{i+1}^p|v_i, T\right]\right)\right] \end{aligned} \tag{5.16}$$

Note that the term $\mathrm{Var}_T\left(\mathbb{E}\left[F_{i+1}^p|v_i,T\right]\right)$ refers to the variance of a $(q+1)$-valued random variable that takes the value $\mathbb{E}_{v_i}\left[F_{i+1}^p|v_i,T=t\right]$ with probability $\Pr\{T=t|v_i\}$ for $t\in\{e,1,2,\cdots,q\}$. Once again, we present a crucial lemma that bounds from below the sum of variances of frequencies with respect to T assuming that the overall advantage is large enough.

Lemma 5.5.11. *For any realization of v_i, the following holds true if $\mathsf{adv}^{q,z}(v_i,A_{r_{i+1}})\geq 3q\varepsilon^2$:*

$$\sum_{p=1}^{q}\mathrm{Var}_T\left(\mathbb{E}\left[F_{i+1}^p|v_i,T\right]\right)\geq\left(\frac{\mathsf{adv}^{q,z}(v_i,A_{r_{i+1}})-3q\varepsilon^2}{2z+1}\right)^2$$

We defer the proof of Lemma 5.5.11 to Section 5.5.6. Using Jensen inequality, the fact that $z\leq q$, and Lemma 5.5.11 along with (5.16) give that

$$f_{i+1}\geq f_i+\mathbb{E}_{v_i}\left[\left(\frac{\mathsf{adv}^{q,z}(v_i,A_{r_{i+1}})-3q\varepsilon^2}{2z+1}\right)^2\right]\geq f_i+\left(\frac{\varepsilon/q-3q\varepsilon^2}{2q+1}\right)^2=f_i+\Omega\left(\frac{\varepsilon^2}{q^4}\right)$$

for sufficiently small $\varepsilon>0$. □

5.5.5 Proof of Theorem 5.1.3

To establish Theorem 5.1.3, we closely follow the concatenation scheme presented in Section 5.4. In the following, we provide a high-level description of the proof skipping the details mentioned in Section 5.4 and highlighting the necessary extra steps.

The construction of the concatenated code is exactly as in Section 5.4 with the exception that the inner code is defined over an alphabet of size q. Note that if $(\gamma,\delta)\in(1-\varepsilon)F_q$, then (γ,δ) lies underneath one of the lines in the set of lines represented by (5.12). In other words, there exists some $z\in\{1,2,\cdots,q-1\}$ for which

$$\gamma+(2z)\delta\leq(1-\varepsilon)\left(\frac{(2q-1)z-z^2}{q}\right).$$

Similar to Section 5.4, we define the notion of *error count* for each block in the codewords of the concatenated code as

$$(I+2z\cdot D)\cdot\frac{q}{(2q-1)z-z^2}$$

where D and I denote the number of deletions and insertions occurred in the block respectively. As in Section 5.4 one can show that at least $\frac{3\varepsilon}{4}\cdot n_{\text{out}}$ of the blocks contain no more than $\left(1-\frac{\varepsilon}{4}\right)n_{\text{in}}$ error count. We denote the set of all such blocks by S. Once again, we partition S into subsets $S_1,S_2,\cdots$ depending on the number of deletions occurred in the set. More precisely, we define $S_i\subseteq S$ as the set of blocks in S that contain a number of deletions that is in the range $\left[n_{\text{in}}\cdot\frac{\varepsilon}{16q}\cdot(i-1),n_{\text{in}}\cdot\frac{\varepsilon}{16q}\cdot i\right)$ for $i=1,2,\cdots,16q/\varepsilon$. Once again, the following hold true:

1. We have that $S = \dot{\bigcup}_{i=1}^{16q/\varepsilon} S_i$. By the Pigeonhole principle, for some $i^* \in [1, 16q/\varepsilon]$, $|S_{i^*}| \geq \frac{3\varepsilon^2}{64q} n_{\text{out}}$.

2. Take some $i \in \{1, 2, \cdots, 16q/\varepsilon\}$ and some block in S_i. Say D deletions have occurred in that block. Then, the total number of insertions is at most $(1-\varepsilon/4)\frac{(2q-1)z-z^2}{q} n_{\text{in}} - 2zD$. Therefore, the total length of the block is

$$\begin{aligned} & n_{\text{in}} - D(1-\varepsilon/4)\frac{(2q-1)z-z^2}{q} n_{\text{in}} - 2zD \\ = \; & n_{\text{in}} \cdot \left[1 + \left(1 - \frac{\varepsilon}{4}\right)\frac{(2q-1)z-z^2}{q}\right] - (2z+1)D \end{aligned} \tag{5.17}$$

which is no more than

$$n_{\text{in}} \cdot \left[1 + \left(1 - \frac{\varepsilon}{4}\right)\frac{(2q-1)z-z^2}{q} - \frac{\varepsilon}{16q}(i-1)(2z+1)\right] \tag{5.18}$$

Based on these observations, it is easy to verify that the decoding algorithm and analysis as presented in Section 5.4 and Algorithm 5 work for the q-ary case with the following minor modifications:

   (a) Based on (5.18), the parameter w determining the length of the window should be

$$w = \left\lfloor \frac{n_{\text{in}} \cdot \left[1 + \left(1 - \frac{\varepsilon}{4}\right)\frac{(2q-1)z-z^2}{q} - \frac{\varepsilon}{16q}(i-1)(2z+1)\right]}{n_{\text{in}}\varepsilon/16} \right\rfloor + 1. \tag{5.19}$$

   (b) As in (5.11), parameter ε_{in} has to be chosen such that the error count in decoding windows does not exceed $n_{\text{in}}(1-\varepsilon_{\text{in}})$. Note that the choice of shifting steps for the decoding window from (5.19) may add up to $\frac{n_{\text{in}}\varepsilon}{16}$ additional insertions to the decoding window. Further, there is up to $n_{\text{in}}\frac{\varepsilon}{16q}$ uncertainty in the total length of the block from (5.17) since $D \in \left[n_{\text{in}} \cdot \frac{\varepsilon}{16q} \cdot (i-1), n_{\text{in}} \cdot \frac{\varepsilon}{16q} \cdot i\right)$. This can also add up to $n_{\text{in}}\frac{\varepsilon}{16q}(2z+1) \leq \frac{\varepsilon}{8}$ insertions. Therefore, we need

$$n_{\text{in}}(1-\varepsilon/4) + n_{\text{in}}\left(\frac{\varepsilon}{16} + \frac{\varepsilon}{8}\right) \cdot \frac{q}{(2q-1)z-z^2} \leq n_{\text{in}}(1-\varepsilon_{\text{in}}).$$

   Note that $\frac{q}{(2q-1)z-z^2} \leq \frac{q}{2q-2} \leq 1$. Hence, it suffuces that $1 - \frac{\varepsilon}{4} + \frac{\varepsilon}{8} + \frac{\varepsilon}{16} \leq 1 - \varepsilon_{\text{in}}$ or equivalently, $\varepsilon_{\text{in}} \leq \frac{\varepsilon}{16}$.

   (c) Some modifications are necessary to the parameters of the outer code. Notably, for alphabet size q, $|S_{i^*}| \geq \frac{3\varepsilon^2}{64q} n_{\text{out}}$ and the fraction of deletions can be as high as $1 - \frac{1}{q}$. This requires $\delta_{\text{out}} = 1 - \frac{3\varepsilon^2}{128q^2}$.

   (d) Finally, note the the value of z is not know to the decoder. So the decoder has to run the algorithm with modifications mentioned above for all possible values of $z = 1, 2, \cdots, q-1$ and the output the union of all lists produced.

5.5.6 Proof of Lemma 5.5.11

We break down this proof into four steps. In the first step, similar to Lemma 5.3.4, we modify v_i and $A_{r_{i+1},n}$ into a simpler structure without significantly changing the advantage. In the second step, we provide an upper bound for the advantage in this modified version that depends on the local frequencies of symbols, more specifically, on what we refer to as $\mathbb{E}\left[F_{i+1}^j | v_i, T=j\right]$. In Step 3, we show that these upper-bounds would yield a non-positive value on the advantage if one replaces the local frequencies with the overall frequency of symbols in v_i, i.e., F_i^j. In the fourth and last step, we show that this means that the local frequencies have to significantly deviate from global ones to attain the advantage achieved by $\bar{M}_i$ (i.e., $\mathsf{adv}_{\bar{M}_i}^{q,z}$), so much that the lower-bound promised in the lemma's statement is achieved.

Step 1. Modifying v_i and $A_{r_{i+1},n}$ for the sake of simplicity: The proof starts with modifying v_i, $A_{r_{i+1},n}$, and the advantage-yielding matching M_i between them in a way that only slightly changes the value of advantage taking steps identical to the one in Lemma 5.3.4. Similar to Lemma 5.3.4, we denote the projection of v_i under M_i by $g = v_i \to M_i$. (See Fig. 5.3 for a depiction of the steps in binary case.)

1. First, we delete all substrings of U_e–i.e., substrings of length l_{i+1} in v_i whose projection does not entirely fall into some stretch of $j^{r_{i+1}}$–from v_i.

2. We reorder the substrings of length l_{i+1} in v_i by shifting all U_j substrings together and the projections in g to preserve the remainder of M_i from step 1.

3. At this point, string g consists of a stretch of symbol 1 followed by a stretch of symbol 2, etc. If the length of all stretches are not equal, we add adequate symbols to each stretch to make g have the form of $1^t 2^t \cdots q^t$.

To track the changes in $\mathsf{adv}_{M_i}^{q,z}$ during this transformation, we track how $|M_i|$, $|v_i|$ and $|g|$ change throughout the three steps mentioned above.

In the first step, a total of up to $|U_e| l_{i+1}$ elements are removed from v_i and M_i. Note that since the run length of $A_{r_{i+1}}$ is r_{i+1}, there can only be $\frac{|g|}{r_{i+1}}$ substrings in U_e. Therefore,

$$|U_e| l_{i+1} \leq \frac{|g| l_{i+1}}{r_{i+1}} = |g|\varepsilon^2 \leq 2\varepsilon^2 |v_i|.$$

The second step preserves $|M_i|$, $|v_i|$ and $|g|$.

Finally, since g is a substring of $A_{r_{i+1}}$, the third step increases $|g|$ only by up to $q r_{i+1}$. Note the run length of the $A_{r_{i+1}}$s and consequently l_{i+1}s are different by a multiplicative factor of at least $\frac{1}{\varepsilon^4}$ by the definition of the code $\mathcal{C}$. Therefore, $q r_{i+1} = \frac{q l_{i+1}}{\varepsilon^2} = \frac{q l_{i+1} |v_i|}{\varepsilon^2 |v_i|} = \frac{q l_{i+1} |v_i|}{\varepsilon^2 l_i} \leq \varepsilon^2 q |v_i|$.

Overall, the value of the $\mathsf{adv}_{M_i}^{q,z} = \frac{(2z+1)|M| - |v_i| - \frac{z+z^2}{q} \cdot |g|}{|v_i|}$ can be affected by a maximum of $2z \times 2\varepsilon^2 |v_i| + q\varepsilon^2 |v_i| = (2z+q)\varepsilon^2 |v_i| \leq 3q\varepsilon^2 |v_i|$ decrease in the numerator and $\varepsilon^2 |v_i|$ decrease

in the denominator. Therefore, the eventual advantage does not drop below $\mathsf{adv}^{q,z}_{M_i} - 3q\varepsilon^2$. Let us denote the transformed versions of v_i, g, and M_i by $\bar{v}_i$, $\bar{g}$, and $\bar{M}_i$ respectively. We have shown that

$$\mathsf{adv}^{q,z}_{\bar{M}_i} \geq \mathsf{adv}^{q,z}_{M_i} - 3q\varepsilon^2. \tag{5.20}$$

Step 2. Bounding Above $\mathsf{adv}^{q,z}_{\bar{M}_i}$ with f^*: Let $\bar{v}_i = (\bar{v}_i^1, \bar{v}_i^2, \cdots, \bar{v}_i^q)$ so that $\bar{v}_i^j$ corresponds to the part of $\bar{v}_i$ that is mapped to j^t under $\bar{M}_i$. Further, let $f_j^* = \mathbb{E}\left[F_{i+1}^j | v_i, T = j\right]$ represent the frequency of the occurrence of symbol j in $\bar{v}_i^j$ as a shorthand, i.e.,

$$f_j^* = \frac{\mathsf{count}_j(\bar{v}_i^j)}{|\bar{v}_i^j|}$$

and p_j be the relative length of $\bar{v}_i^j$, i.e.,

$$p_j = \frac{|\bar{v}_i^j|}{|\bar{v}_i|}.$$

In this section, we compute an upperbound for $\mathsf{adv}^{q,z}_{\bar{M}_i}$ that depends on f_j^*s. For the sake of simplicity, from now on we assume, without loss of generality, that

$$\mathsf{count}_1(\bar{v}_i^1) \geq \mathsf{count}_2(\bar{v}_i^2) \geq \cdots \geq \mathsf{count}_q(\bar{v}_i^q)$$

or equivalently,

$$f_1^* p_1 \geq f_2^* p_2 \geq \cdots \geq f_q^* p_q.$$

Consider the matching between $\bar{v}_i$ and $\bar{p}$ that, for any $j \in \{1, 2, \cdots, q\}$ matches as many js as possible from j^t to $\bar{v}_i^j$. This matching clearly yields the largest possible advantage between the two that is an upperbound for the $\mathsf{adv}^{q,z}_{\bar{M}_i}$. Similar to the binary case, we find a t that maximizes this advantage and use its advantage as an upper-bound for $\mathsf{adv}^{q,z}_{\bar{M}_i}$.

Let c be so that $f_c^*|\bar{v}_i^c| > t \geq f_{c+1}^*|\bar{v}_i^{c+1}|$. Then, increasing t by one would increase the length of $\bar{p}$ by q and increases the size of the matching by c. To see the effect of this increment on the advantage, note that the denominator does not change and the numerator changes by $c(2z+1) - \frac{z+z^2}{q} \cdot q$. This change in advantage is positive as long as

$$\begin{aligned} & c(2z+1) - (z+z^2) \geq 0 \\ \Leftrightarrow \quad & c \geq \frac{z+z^2}{2z+1} = \frac{z}{2} + \left(\frac{1}{4} - \frac{1}{4(2z+1)}\right). \end{aligned}$$

Note that the term $\frac{1}{4} - \frac{1}{4(2z+1)}$ is always between $\left[0, \frac{1}{4}\right]$. Hence, incrementing t increases the advantage as long as $c \geq \lfloor \frac{z}{2} \rfloor + 1$. This means that the highest possible advantage is derived when $t = f_w^*|\bar{v}_i^w|$ for $w = \lfloor \frac{z}{2} \rfloor + 1$. With this value for t, the matching contains $f_j^*|\bar{v}_i^j|$ edges between j^t and $|\bar{v}_i^j|$ for all $j > w$ and t edges between j^t and $|\bar{v}_i^j|$ for $j \leq w$. Therefore, the size of this matching is

$$tw + \sum_{j=w+1}^{q} f_j^*|\bar{v}_i^j|.$$

This yields the following advantage

$$\begin{aligned}
&\frac{(2z+1)\left[tw+\sum_{j=w+1}^{q} f_j^*|\bar{v}_i^j|\right]-|\bar{v}_i|-\frac{z+z^2}{q}\cdot qt}{|\bar{v}_i|}\\
=\;&\frac{(2z+1)\left[f_w^*|\bar{v}_i^w|w+\sum_{j=w+1}^{q} f_j^*|\bar{v}_i^j|\right]-|\bar{v}_i|-\frac{z+z^2}{q}\cdot qf_w^*|\bar{v}_i^w|}{|\bar{v}_i|}\\
=\;&(2z+1)\left[f_w^*p_w w+\sum_{j=w+1}^{q} f_j^*p_j\right]-1-(z+z^2)\cdot f_w^*p_w\\
=\;&\left[(2z+1)w-(z+z^2)\right]\cdot f_w^*p_w+(2z+1)\sum_{j=w+1}^{q} f_j^*p_j-1
\end{aligned}$$

We remind that this is an upper-bound on the $\mathsf{adv}_{\bar{M}_i}^{q,z}$. Next, we plug in $w=\lfloor\frac{z}{2}\rfloor+1$ into this bound. Note that

$$(2z+1)w-(z+z^2)=z(2w-z)+w-z=\begin{cases}\frac{3z+2}{2} & \text{If } z \text{ is even}\\ \frac{z+1}{2} & \text{If } z \text{ is odd}\end{cases}$$

Therefore, we have the following set of upper-bounds on the advantage

$$\mathsf{adv}_{\bar{M}_i}^{q,z}\le\tfrac{3z+2}{2}\cdot f_w^*p_w+(2z+1)\textstyle\sum_{j=w+1}^{q} f_j^*p_j-1 \quad \text{If } z \text{ is even} \tag{5.21}$$

$$\mathsf{adv}_{\bar{M}_i}^{q,z}\le\tfrac{z+1}{2}\cdot f_w^*p_w+(2z+1)\textstyle\sum_{j=w+1}^{q} f_j^*p_j-1 \quad \text{If } z \text{ is odd} \tag{5.22}$$

Step 3. Proving Non-positivity of the Bound from Step 3 for Unit Sum Vectors: In this step, we show that the bounds (5.21) and (5.22) on advantage that were presented in Step 2 are necessarily non-positive for any vector $(f_1^*,\cdots,f_q^*)$ with unit sum including the vector of overall frequencies $\bar{f}=(\bar{f}_1,\cdots,\bar{f}_q)$ where $\bar{f}_j=\frac{\mathsf{count}_j(\bar{v}_i)}{|\bar{v}_i|}=F_i^j$. In Step 4, we use this fact to show that f^* needs to deviate noticeably from $\bar{f}$ which gives that the variance of frequencies with respect to T is large enough, thus finishing the proof.

Proposition 5.5.12. *Let $(p_1,\cdots,p_q)$ and $(f_1^*,\cdots,f_q^*)$ be two positive real vectors with unit sum that satisfy*

$$f_1^*p_1\ge f_2^*p_2\ge\cdots\ge f_q^*p_q.$$

Then, for all integers $1\le z<q$, the following hold for $w=\lfloor\frac{z}{2}\rfloor+1$:

1. *If z is even,*
$$\frac{3z+2}{2}\cdot f_w^*p_w+(2z+1)\sum_{j=w+1}^{q} f_j^*p_j\le 1.$$
2. *If z is odd,*
$$\frac{z+1}{2}\cdot f_w^*p_w+(2z+1)\sum_{j=w+1}^{q} f_j^*p_j\le 1.$$

We defer the proof of Proposition 5.5.12 to Section 5.5.7.

Step 4. Large Deviation of f^*s from $\bar{f}$s and Large Variance: Here we finish the proof assuming z is odd. The even case can be proved in the same way. Note that Proposition 5.5.12 gives that for the overall frequency vector $\bar{f}$ which has a unit sum,

$$\frac{z+1}{2} \cdot \bar{f}_w p_w + (2z+1) \sum_{j=w+1}^{q} \bar{f}_j p_j - 1 \leq 0. \tag{5.23}$$

However, (5.20) and (5.22) imply that for local frequency vector f^*

$$\frac{z+1}{2} \cdot f^*_w p_w + (2z+1) \sum_{j=w+1}^{q} f^*_j p_j - 1 \geq \mathsf{adv}^{q,z}_{M_i} - 3q\varepsilon^2. \tag{5.24}$$

Subtracting (5.23) from (5.24) gives that

$$\begin{aligned}
&\frac{z+1}{2} \cdot p_w(f^*_w - \bar{f}_w) + (2z+1) \sum_{j=w+1}^{q} (f^*_j - \bar{f}_j)p_j \geq \mathsf{adv}^{q,z}_{M_i} - 3q\varepsilon^2. \\
\Rightarrow\ &\frac{z+1}{2} \cdot p_w|f^*_w - \bar{f}_w| + (2z+1) \sum_{j=w+1}^{q} |f^*_j - \bar{f}_j|p_j \geq \mathsf{adv}^{q,z}_{M_i} - 3q\varepsilon^2. \\
\Rightarrow\ &(2z+1) \sum_{j=w}^{q} |f^*_j - \bar{f}_j|p_j \geq \mathsf{adv}^{q,z}_{M_i} - 3q\varepsilon^2. \\
\Rightarrow\ &\sum_{j=w}^{q} |f^*_j - \bar{f}_j|p_j \geq \frac{\mathsf{adv}^{q,z}_{M_i} - 3q\varepsilon^2}{2z+1}.
\end{aligned}$$

This means that there exists some j_0 for which

$$|f^*_{j_0} - \bar{f}_{j_0}|p_{j_0} \geq \frac{\mathsf{adv}^{q,z}_{M_i} - 3q\varepsilon^2}{2z+1} \Rightarrow (f^*_{j_0} - \bar{f}_{j_0})^2 p_{j_0} \geq (f^*_{j_0} - \bar{f}_{j_0})^2 p^2_{j_0} \geq \left(\frac{\mathsf{adv}^{q,z}_{M_i} - 3q\varepsilon^2}{2z+1}\right)^2.$$

Note that

$$\begin{aligned}
\sum_{p=1}^{q} \mathrm{Var}_T\left(\mathbb{E}\left[F^p_{i+1}|v_i, T\right]\right) &= \sum_{p=1}^{q}\sum_{j=1}^{q} \left(\mathbb{E}\left[F^p_{i+1}|v_i, T=j\right] - F^p_i\right)^2 \Pr\{T=j|v_i\} \\
&\geq \left(\mathbb{E}\left[F^{j_0}_{i+1}|v_i, T=j_0\right] - F^{j_0}_i\right)^2 \Pr\{T=j_0|v_i\} \\
&= (f^*_{j_0} - \bar{f}_{j_0})^2 p_{j_0} \geq \left(\frac{\mathsf{adv}^{q,z}_{M_i} - 3q\varepsilon^2}{2z+1}\right)^2.
\end{aligned}$$

□

5.5.7 Proof of Proposition 5.5.12

To prove Proposition 5.5.12 we provide several observations that simplify the form of the solution that yields the maximum value by reducing the number of important free variables.

Observation 5.5.13. *Any solution that maximizes the left-hand-side satisfies*

$$f_1^* p_1 = f_2^* p_2 = \cdots = f_w^* p_w.$$

We start with $f_{w-1}^* p_{w-1} = f_w^* p_w$. Assume by contradiction that $f_{w-1}^* p_{w-1} > f_w^* p_w$. Then, there exists a small positive value ϵ for which decreasing f_{w-1}^* by ϵ and increasing f_w^* by ϵ would preserve $f_{w-1}^* p_{w-1} \geq f_w^* p_w$ but increase the overall value of the expression. This contradicts the fact that the solution maximizes the left-hand-side value. Similarly, if $f_{w-2}^* p_{w-2} > f_{w-1}^* p_{w-1} = f_w^* p_w$, same idea executed on $f_{w-2}^* p_{w-2}$ and $f_{w-1}^* p_{w-1}$ turns the solution into one for which $f_{w-2}^* p_{w-2} \geq f_{w-1}^* p_{w-1} > f_w^* p_w$ which is, again, contradictory to the fact that the solution maximizes the left-hand-size. Continuing this argument gives Observation 5.5.13.

We next present the two following lemmas that we will prove later in Section 5.5.7.

Lemma 5.5.14. *Let $f_1, \cdots, f_q$ and $p_1, \cdots, p_q$ be positive numbers for which $\sum_{i=1}^q f_i = F$, $\sum_{i=1}^q p_i = P$ and $f_1 p_1 \geq f_2 p_2 \geq \cdots \geq f_q p_q$. Then*

$$f_q p_q \leq \frac{FP}{q^2}$$

and equality is attained only at $f_i = \frac{F}{q}$ and $p_i = \frac{P}{q}$ for all $i \in \{1, 2, \cdots, q\}$.

Lemma 5.5.15. *Let $f_1, \cdots, f_q$ and $p_1, \cdots, p_q$ be positive variables with constraints $\sum_{i=1}^q f_i = F$, $\sum_{i=1}^q p_i = P$, $f_1 p_1 \geq f_2 p_2 \geq \cdots \geq f_q p_q$, and $f_1 p_1 \leq m$ for some constant m. Then, the largest possible value for $\sum_{i=1}^q f_i p_i$ is:*

$$f_{\max}(F, P, m) = \begin{cases} FP & \text{if } FP \leq m \\ um + (\sqrt{FP} - u\sqrt{m})^2 & \text{if } \frac{FP}{(u+1)^2} \leq m < \frac{FP}{u^2} \text{ for } u = 1, 2, \cdots, q-1 \\ mq & \text{if } m < \frac{FP}{q^2} \end{cases}$$

We claim that if one fixes the two quantities $f_w^* p_w = \alpha$ and $\sum_{j=1}^w p_j = \beta$, then using observation 1 and Lemmas 5.5.14 and 5.5.15, the maximum value of the two terms in the statement of the theorem can be written in terms of α and β. Note that with $f_w^* p_w = \alpha$, both expressions are maximized when $\sum_{j=w+1}^q f_j^* p_j$ is maximized and according to Lemma 5.5.15, that happens when $(\sum_{i=w+1}^q f_i^*)(\sum_{i=w+1}^q p_i) = (\sum_{i=w+1}^q f_i^*)(1-\beta)$ is maximized or equivalently $\sum_{i=1}^w f_i^*$ is as small as possible.

Now, note that for $j \leq w$ all $f_j^* p_j$'s are larger than or equal to α. Then according to Lemma 5.5.14, $\frac{(\sum_{i=1}^w f_i^*) \times \beta}{w^2} \geq \alpha \Rightarrow \sum_{i=1}^w f_i^* \geq \frac{\alpha w^2}{\beta}$.

All in all, the above-mentioned observations and lemmas boil down the two parts of theorem statement to the following:

For any $\alpha, \beta \in [0, 1]$ where $\frac{\alpha w^2}{\beta} \leq 1$:

1. If z is even,

$$\frac{3z+2}{2}\alpha + (2z+1) f_{\max}\left(1 - \frac{\alpha w^2}{\beta}, 1-\beta, \alpha\right) \leq 1$$

2. If z is odd,

$$\frac{z+1}{2}\alpha + (2z+1) f_{\max}\left(1-\frac{\alpha w^2}{\beta}, 1-\beta, \alpha\right) \leq 1$$

Note that to maximize $f_{\max}$ term for a given α, one needs to maximize $\left(1-\frac{\alpha w^2}{\beta}\right)(1-\beta)$. This is attained with the following choice of $\beta = \sqrt{\alpha}w$. With this choice of β we have

$$\begin{aligned} f_{\max}\left(1-\sqrt{\alpha}w, 1-\sqrt{\alpha}w, \alpha\right) &= \begin{cases} (1-\sqrt{\alpha}w)^2 & \text{if } (1-\sqrt{\alpha}w)^2 \leq \alpha \\ u\alpha + (1-w\sqrt{\alpha}-u\sqrt{\alpha})^2 & \text{if } \frac{(1-\sqrt{\alpha}w)^2}{(u+1)^2} \leq \alpha < \frac{(1-\sqrt{\alpha}w)^2}{u^2} \\ & \text{for } 1 \leq u \leq q-w \\ \alpha q & \text{if } \alpha < \frac{(1-\sqrt{\alpha}w)^2}{(q-w)^2} \end{cases} \\ &= \begin{cases} (1-\sqrt{\alpha}w)^2 & \text{if } \alpha \in \left[\frac{1}{(w+1)^2}, \frac{1}{w^2}\right] \\ u\alpha + (1-(w+u)\sqrt{\alpha})^2 & \text{if } \alpha \in \left[\frac{1}{(w+u+1)^2}, \frac{1}{(w+u)^2}\right) \\ & \text{for } 1 \leq u \leq q-w \\ \alpha q & \text{if } \alpha < \frac{1}{q^2} \end{cases} \end{aligned}$$

Note that we require that $\beta \leq 1 \Rightarrow \alpha \leq \frac{1}{w^2}$. Therefore in the second line the regions for α are truncated at $\frac{1}{w^2}$.

As the next step, we plug in the above description for $f_{\max}$ into each of the two terms and derive a piece-wise characterization of them based on α.

1. If z is even,

$$LHS = \begin{cases} \frac{3z+2}{2}\alpha + (2z+1)(1-\sqrt{\alpha}w)^2 & \text{if } \alpha \in \left[\frac{1}{(w+1)^2}, \frac{1}{w^2}\right] \\ \frac{3z+2}{2}\alpha + (2z+1)\left[u\alpha + (1-(u+w)\sqrt{\alpha})^2\right] & \text{if } \alpha \in \left[\frac{1}{(w+u+1)^2}, \frac{1}{(w+u)^2}\right) \\ & \text{for } u = 1, 2, \cdots, q-w \\ \frac{3z+2}{2}\alpha + (2z+1)\alpha q & \text{if } \alpha < \frac{1}{q^2} \end{cases}$$

Note that this function is continuous. The derivative in $\alpha < \frac{1}{q^2}$ region is positive meaning that the function is increasing in that region.

For the region $\alpha \in \left[\frac{1}{(w+1)^2}, \frac{1}{w^2}\right]$,

$$\frac{\partial^2}{\partial \alpha^2}\left[\frac{3z+2}{2}\alpha + (2z+1)(1-\sqrt{\alpha}w)^2\right] = (z/2+1)(z+1/2)\alpha^{-3/2} > 0$$

Therefore, the function is concave in this region; giving that the maximum value in this region is obtained either at $\frac{1}{(w+1)^2}$ or $\frac{1}{w^2}$. Note that we can easily exclude $\frac{1}{w^2}$ as LHS function has a value of zero there.

We now analyze the derivative for the regions of form $\alpha \in \left[\frac{1}{(w+u+1)^2}, \frac{1}{(w+u)^2}\right]$

$$\frac{\partial}{\partial \alpha}\left[\frac{3z+2}{2}\alpha + (2z+1)\left(u\alpha + (1-(u+w)\sqrt{\alpha})^2\right)\right]$$
$$= \frac{3z+2+(4z+2)((u+w)^2+u)}{2} - \frac{(u+w)(2z+1)}{\sqrt{\alpha}}$$

and hence,

$$\frac{\partial^2}{\partial \alpha^2}\left[\frac{3z+2}{2}\alpha + (2z+1)\left(u\alpha + (1-(u+w)\sqrt{\alpha})^2\right)\right] = (u+w)(2z+1)\alpha^{-3/2}$$

and is always positive. Giving that within each region of form $\alpha \in \left[\frac{1}{(w+u+1)^2}, \frac{1}{(w+u)^2}\right]$ the expression is concave and attains no local maximum. The above observations along with the fact that this piece-wise function is continuous, gives that the global maximum is necessarily of the form $\alpha = \frac{1}{(w+u+1)^2}$ for some $u = 0, 1, 2, \cdots, q-w$. Note that at such point the value of LHS is

$$\begin{aligned} LHS(u) &= \frac{3z+2}{2}\alpha + (2z+1)\left[u\alpha + (1-(w+u)\sqrt{\alpha})^2\right]\Big|_{\alpha=\frac{1}{(w+u+1)^2}} \\ &= \frac{3z+2}{2(w+u+1)^2} + (2z+1)\frac{u+1}{(w+u+1)^2} \\ &= \frac{3z+2+2(2z+1)(u+1)}{2(w+u+1)^2} = \frac{7z+4+2(2z+1)u}{2(w+u+1)^2} \end{aligned}$$

To find the optimum u, we take derivative with respect to u.

$$\begin{aligned} & \frac{\partial}{\partial u} LHS(u) = 0 \\ \Leftrightarrow\ & 2(2z+1)2(w+u+1)^2 - (7z+4+2(2z+1)u)4(w+u+1) = 0 \\ \Leftrightarrow\ & (2z+1)(w+u+1) - (7z+4+2(2z+1)u) = 0 \\ \Leftrightarrow\ & -u(2z+1) + (2z+1)(z/2+2) - (7z+4) = 0 \\ \Leftrightarrow\ & u = \frac{(2z+1)(z/2+2)-(7z+4)}{2z+1} = \frac{z^2+\frac{9}{2}z+2-7z-4}{2z+1} \\ \Leftrightarrow\ & u = \frac{(2z+1)(z/2+2)-(7z+4)}{2z+1} = \frac{z^2-\frac{5}{2}z-2}{2z+1} = \frac{z}{2} - \frac{3z+2}{2z+1} \end{aligned}$$

Note that the term $\frac{3z+2}{2z+1}$ is always between 1 and 2. Hence, the maximum is achieved either at $u = \frac{z}{2} - 1$ or $u = \frac{z}{2} - 2$. We simply compute $LHS(u)$ for both of these values to obtain the maximum.

$$LHS\left(\frac{z}{2}-1\right)=\frac{7z+4+2(2z+1)(z/2-1)}{2(z+1)^2}=\frac{2z^2+4z+2}{2(z+1)^2}=1$$

and

$$LHS\left(\frac{z}{2}-2\right)=\frac{7z+4+2(2z+1)(z/2-2)}{2z^2}=\frac{2z^2}{2z^2}=1$$

meaning that, indeed, the maximum achievable value for even z is 1. This finishes the proof for even zs. The maximum value 1 can be achieved by $f_1^*=\cdots=f_m^*=\frac{1}{m}=p_1=\cdots=p_m$ and all other values equal to zero for $m=z$ or $z+1$.

2. If z is odd,

$$LHS=\begin{cases}\frac{z+1}{2}\alpha+(2z+1)(1-\sqrt{\alpha}w)^2 & \text{if } \alpha\in\left[\frac{1}{(w+1)^2},\frac{1}{w^2}\right]\\ \frac{z+1}{2}\alpha+(2z+1)\left[u\alpha+(1-(u+w)\sqrt{\alpha})^2\right] & \text{if } \alpha\in\left[\frac{1}{(w+u+1)^2},\frac{1}{(w+u)^2}\right)\\ & \text{for } u=1,2,\cdots,q-w\\ \frac{z+1}{2}\alpha+(2z+1)\alpha q & \text{if } \alpha<\frac{1}{q^2}\end{cases}$$

Note that this function is continuous. The derivative in $\alpha<\frac{1}{q^2}$ region is positive meaning that the function is increasing in that region.

Similar to the even z case, for regions $\alpha\in\left[\frac{1}{(w+1)^2},\frac{1}{w^2}\right]$ and $\alpha\in\left[\frac{1}{(w+u+1)^2},\frac{1}{(w+u)^2}\right]$, the second derivative is positive.

$$\frac{\partial^2}{\partial\alpha^2}\left[\frac{z+1}{2}\alpha+(2z+1)(1-\sqrt{\alpha}w)^2\right]=\frac{(z+1)(z+1/2)}{2}\alpha^{-3/2}>0$$

$$\frac{\partial^2}{\partial\alpha^2}\left[\frac{z+1}{2}\alpha+(2z+1)\left(u\alpha+(1-(u+w)\sqrt{\alpha})^2\right)\right]=(u+w)(2z+1)\alpha^{-3/2}$$

Meaning that, once again, the global maximum is attained at a point necessarily of the form $\alpha=\frac{1}{(w+u+1)^2}$ for some $u=0,1,2,\cdots,q-w$. Note that at such point the value of LHS is

$$\begin{aligned}LHS(u) &= \frac{z+1}{2}\alpha+(2z+1)\left[u\alpha+(1-(w+u)\sqrt{\alpha})^2\right]\Big|_{\alpha=\frac{1}{(w+u+1)^2}}\\ &= \frac{z+1+2(2z+1)(u+1)}{2(w+u+1)^2}=\frac{5z+3+2(2z+1)u}{2(w+u+1)^2}\end{aligned}$$

To find the optimum u, we take derivative with respect to u.

$$\begin{aligned}
& \frac{\partial}{\partial u} LHS(u) = 0 \\
\Leftrightarrow \quad & 2(2z+1)2(w+u+1)^2 - (5z+3+2(2z+1)u)4(w+u+1) = 0 \\
\Leftrightarrow \quad & (2z+1)(w+u+1) - (5z+3+2(2z+1)u) = 0 \\
\Leftrightarrow \quad & -u(2z+1) + (2z+1)\frac{z+3}{2} - (5z+3) = 0 \\
\Leftrightarrow \quad & u = \frac{(2z+1)(z+3)/2 - (5z+3)}{2z+1} = \frac{z^2 + \frac{7}{2}z + 3/2 - 5z - 3}{2z+1} \\
\Leftrightarrow \quad & u = \frac{z^2 - \frac{3}{2}z - \frac{3}{2}}{2z+1} = \frac{z-1}{2} - \frac{z+1}{2z+1}
\end{aligned}$$

Note that the term $\frac{z+1}{2z+1}$ is always between 0 and 1. Hence, the maximum is achieved either at $u = \frac{z-1}{2}$ or $u = \frac{z-3}{2}$. We simply compute $LHS(u)$ for both of these values to obtain the maximum.

$$LHS\left(\frac{z-1}{2}\right) = \frac{5z+3+2(2z+1)(z-1)/2}{2(z+1)^2} = \frac{2z^2+4z+2}{2(z+1)^2} = \frac{z^2+2z+1}{z^2+2z+1} = 1$$

and

$$LHS\left(\frac{z-3}{2}\right) = \frac{5z+3+2(2z+1)(z-3)/2}{2z^2} = \frac{2z^2}{2z^2} = 1$$

meaning that, indeed, the maximum achievable value for odd z is 1. This finishes the proof. The maximum value 1 in the case of odd z can be achieved by setting $f_1^* = \cdots = f_m^* = \frac{1}{m} = p_1 = \cdots = p_m$ and all other values equal to zero for $m = z$ or $z+1$. □

Proof of Auxiliary Lemmas 5.5.14 and 5.5.15

Proof of Lemma 5.5.14. We prove this by induction on q. For the base case of $q = 1$ correctness is trivial. For any $q > 1$, we want to find the f_q and p_q that maximize $f_q p_q$ and for which an appropriate $f_1, \cdots, f_{q-1}$ and $p_1, \cdots, p_{q-1}$ exists. Note that $\sum_{i=1}^{q-1} f_i = 1 - f_q$ and $\sum_{i=1}^{q-1} p_i = 1 - p_q$. Therefore, by the induction hypothesis, the largest possible amount that $f_{q-1}p_{q-1}$ can take would be $\frac{(F-f_q)(P-p_q)}{(q-1)^2}$. This gives that a pair (f_q, p_q) are feasible in equations described in the lemma's statement if and only if $f_q p_q \leq \frac{(F-f_q)(P-p_q)}{(q-1)^2}$.

Note that

$$\begin{aligned}
& f_q p_q \leq \frac{(F-f_q)(P-p_q)}{(q-1)^2} \\
\Rightarrow \quad & f_q \left(p_q + \frac{P-p_q}{(q-1)^2} \right) \leq \frac{F(P-p_q)}{(q-1)^2} \\
\Rightarrow \quad & f_q \leq \frac{F(P-p_q)}{p_q(q-1)^2 + P - p_q} \\
\Rightarrow \quad & f_q p_q \leq \frac{F(P-p_q)p_q}{p_q(q^2-2q)+P}
\end{aligned}$$

We know determine the maximum value of the right hand side over the choice of p_q by setting the derivative to zero.

$$\begin{aligned}
& \frac{(FP - 2Fp_q)(p_q(q^2-2q)+P) - F(P-p_q)p_q(q^2-2q)}{(p_q(q^2-2q)+P)^2} = 0 \\
\Rightarrow \quad & P^2 - 2Pp_q - (q^2-2q)p_q^2 = 0 \\
\Rightarrow \quad & p_q = \frac{-2P \pm \sqrt{4P^2 + 4P^2(q^2-2q)}}{2(q^2-2q)} = \frac{-P \pm \sqrt{P^2(q^2-2q+1)}}{q^2-2q} \\
& = \frac{-P \pm P(q-1)}{q^2-2q}
\end{aligned}$$

The only positive solution is $p_q = \frac{P}{q}$ that yields $f_q p_q = \frac{FP}{q^2}$ with $f_q = \frac{F}{q}$. Note that by the induction hypothesis, this is obtained only when $p_i = \frac{P-p_q}{q-1} = \frac{P}{q}$ and $f_i = \frac{F-f_q}{q-1} = \frac{F}{q}$ for all $i = 1, 2, \cdots, q-1$. □

Proof of Lemma 5.5.15. We start with the simple observation that in any optimal solution in which $f_1 p_1 = \min(m, FP)$. Assume for the sake of contradiction that this is not the case. Let j be the smallest integer such that $f_j p_j > 0$. Clearly, either $f_1 > f_j$ or $p_1 > p_j$. Without loss of generality assume that the former holds. Then, it is easy to verify that there exists a small enough $\varepsilon > 0$ such that reducing f_j by ε and increasing f_1 by ε yields a strictly larger solution and contradicts the optimality assumption.

Having this observation, we prove the lemma by induction over q. As the basis of the induction, take the case where $q = 2$. If $m \geq PQ$, then using the above-mentioned observation, setting $f_1 = F$, $p_1 = P$, and the rest of the variables to zero yields the optimal solution. Otherwise, the observation rules that f_1 and p_1 must be chosen such that $f_1 p_1 = m$. A straight forward calculation shows that with the following choice of f_1 and p_1, $f_1 p_1 = f_2 p_2 = m$ that is trivially an optimal solution.

$$f_1 = \frac{FP + \sqrt{F^2P^2 - 4mFP}}{2P}, \quad p_1 = \frac{FP - \sqrt{F^2P^2 - 4mFP}}{2F}$$

$$f_2 = \frac{FP - \sqrt{F^2P^2 - 4mFP}}{2P}, \quad p_2 = \frac{FP + \sqrt{F^2P^2 - 4mFP}}{2F}$$

For the induction step, assume that the lemma holds for $q-1$. Once again we use the observation to determine f_1 and p_1 first. If $FP \leq m$, setting $f_1 = F$, $p_1 = P$, and all other values to zero gives the optimal solution. Otherwise, we have to choose f_1 and p_1 such that $f_1 p_1 = m$. We can use the induction hypothesis for $q' = q-1$ to set the rest of the variables with parameters $m' = m$, $F' = F - f_1$, and $P' = P - p_1$. Note that $f_{\max}$ is actually a function of FP and not F and P. Therefore, in the optimal solution f_1 and p_1 are chosen such that $f_1 p_1 = m$ and $(F-f_1)(P-p_1) = FP + m - f_1 P - p_1 F$ is maximized, or equivalently, $f_1 P + p_1 F$ is minimized. Note that $f_1 P + p_1 F = f_1 P + \frac{mF}{f_1}$. Hence one has to choose $f_1 = \sqrt{\frac{mF}{P}}$ and $p_1 = \sqrt{\frac{mP}{F}}$.

With this choice for f_1 and p_1, $F'P' = FP + m - 2\sqrt{mFP} = (\sqrt{FP} - \sqrt{m})^2$. Note that if $m < \frac{FP}{u^2} \Leftrightarrow u^2 < \frac{FP}{m} \Leftrightarrow u < \frac{\sqrt{FP}}{\sqrt{m}} = \frac{\sqrt{F'P'}}{\sqrt{m}} + 1 \Leftrightarrow m < \frac{F'P'}{(u-1)^2}$.

Hence, if $\frac{FP}{(u+1)^2} \leq m < \frac{FP}{u^2}$ for some $u = 2, \cdots, q-2$, then $\frac{F'P'}{u^2} \leq m < \frac{F'P'}{(u-1)^2}$ and $f_{\max}(F,P,m) = m + (u-1)m + (\sqrt{F'P'} - (u-1)\sqrt{m})^2 = um + (\sqrt{FP} - m)^2$.

If $\frac{FP}{4} \leq m < FP$, $f_{\max} = f_1 p_1 + f_2 p_2 = m + (F-f_1)(P-p_1) = m + (\sqrt{FP} - \sqrt{m})^2$.

Finally, if $m < \frac{FP}{q^2} \Leftrightarrow m < \frac{(\sqrt{F'P'}+\sqrt{m})^2}{q^2} \Leftrightarrow m < \frac{F'P'}{(q-1)^2}$ and, therefore, $f_{\max} = m + m(q-1) = mq$. □

Chapter 6

Maximum Achievable Rate of List-Decodable Synchronization Codes

In this chapter, we provide results that further complete the picture portrayed by the results of Chapters 4 and 5 in regard to list-decodable insdel codes. We prove several bounds on the list-decoding capacity of worst-case synchronization channels, i.e., the highest rate that is achievable for q-ary list-decodable insdel codes that can correct from δ fraction of deletions and γ fraction of insertions. We present upper-bounds and lower-bounds for the capacity for the cases of insertion-only channels ($\delta = 0$), deletion-only channels ($\gamma = 0$), and the generalized case of channels with both insertions and deletions. Our lower-bounds are derived by analysis of random codes.

Note that this question generalizes the questions that Chapters 4 and 5 answer. Chapter 4 found the maximal achievable rate for codes that can correct (γ, δ) fraction of insdel errors while allowing the alphabet size to be sufficiently large (therefore, ignoring the alphabet size) and Chapter 5 pinned down the resilience region, i.e., the set of (γ, δ) error fractions for which capacity is non-zero. These are both special cases of the question of interest in this chapter.

The results of this chapter also give interesting implications on the code constructions from Chapters 3 and 4.We show that the alphabet size of insdel codes needs to be exponentially large in ε^{-1}, where ε is the gap to capacity above. Our result even applies to settings where the unique-decoding capacity equals the list-decoding capacity and when it does so, it shows that the alphabet size needs to be exponentially large in the gap to capacity. This is sharp contrast to the Hamming error model where alphabet size polynomial in ε^{-1} suffices for unique decoding. This lower bound also shows that the exponential dependence on the alphabet size in Chapter 3 is actually necessary.

6.1 Our Results

The main focus of this chapter is on understanding how the size of the alphabet factors in to the highest rate that an insdel code can achieve. To this end, we provide several bounds that characterize the relation between fundamental characteristics of insdel codes. Including the two extreme cases with only deletions (i.e., $\gamma = 0$) and with only insertions (i.e., with $\delta = 0$). Let us start with an overview the results and bounds that are presented in this chapter.

6.1.1 Upper-Bounds for Rate and Implications on Alphabet Size

We start with the insertion-only setting. We note here that one cannot hope to find a constant rate family of codes that can protect n symbols out of an alphabet of size q against $(q-1)n$ many insertions or more. This is so since, with $(q-1)n$ insertions, one can turn any string $y \in [1..q]^n$ into the fixed sequence $1, 2, \cdots, q, 1, 2, \cdots, q, \cdots, 1, 2, \cdots, q$ by simply inserting $q-1$ many symbols around each symbol of y to construct a $1, \cdots, q$ there. Hence, Theorem 6.1.1 only focuses on codes that protect n rounds of communication over an alphabet of size q against γn insertions for $\gamma < q - 1$.

Theorem 6.1.1. *Any list-decodable family of codes $\mathcal{C}$ that protects against γ fraction of insertions for some $\gamma < q-1$ and guarantee polynomially-large list size in terms of block length cannot achieve a rate R that is strictly larger than $1-\log_q(\gamma+1)-\gamma\left(\log_q \frac{\gamma+1}{\gamma} - \log_q \frac{q}{q-1}\right)$.*

In particular, the theorem asserts that if the code has rate $R = 1-\varepsilon$, then its alphabet size must be exponentially large in $1/\varepsilon$, namely, $q \geq (\gamma+1)^{1/\varepsilon}$.

Next, we turn to the deletion-only case. Here again we note that no constant rate q-ary code can protect against $\delta \geq \frac{q-1}{q}$ fraction of deletions since such a large fraction of deletions may remove all but the most frequent symbol of codewords. Therefore, Theorem 6.1.2 below only concerns codes that protect against $\delta \leq \frac{q-1}{q}$ fraction of deletions.

Theorem 6.1.2. *Any list-decodable family of insdel codes that protect against δ-fraction of deletions (and no insertions) for some $0 \leq \delta < \frac{q-1}{q}$ that are list-decodable with polynomially-bounded list size has rate R upper bounded as below:*

- $R \leq f(\delta) \triangleq (1-\delta)\left(1-\log_q \frac{1}{1-\delta}\right)$ *where $\delta = \frac{d}{q}$ for some integer d.*
- $R \leq (1-q\delta')f\left(\frac{d}{q}\right) + q\delta' f\left(\frac{d+1}{q}\right)$ *where $\delta = \frac{d}{q} + \delta'$ for some integer d and $0 \leq \delta' < \frac{1}{q}$.*

In particular if $\delta = d/q$ for integer d and rate is $1-\delta-\varepsilon$ then the theorem above asserts that $q \geq \left(\frac{1}{1-\delta}\right)^{\frac{1-\delta}{\varepsilon}}$, or in other words q must be exponentially large in $1/\varepsilon$. Indeed such a statement is true for all δ as asserted in the corollary below.

Corollary 6.1.3. *There exists a function $f : (0,1) \to (0,1)$ such that any family of insdel codes that protects against δ-fraction of deletions with polynomially bounded list sizes and has rate $1-\delta-\varepsilon$ must have alphabet size $q \geq \exp\left(\frac{f(\delta)}{\varepsilon}\right)$.*

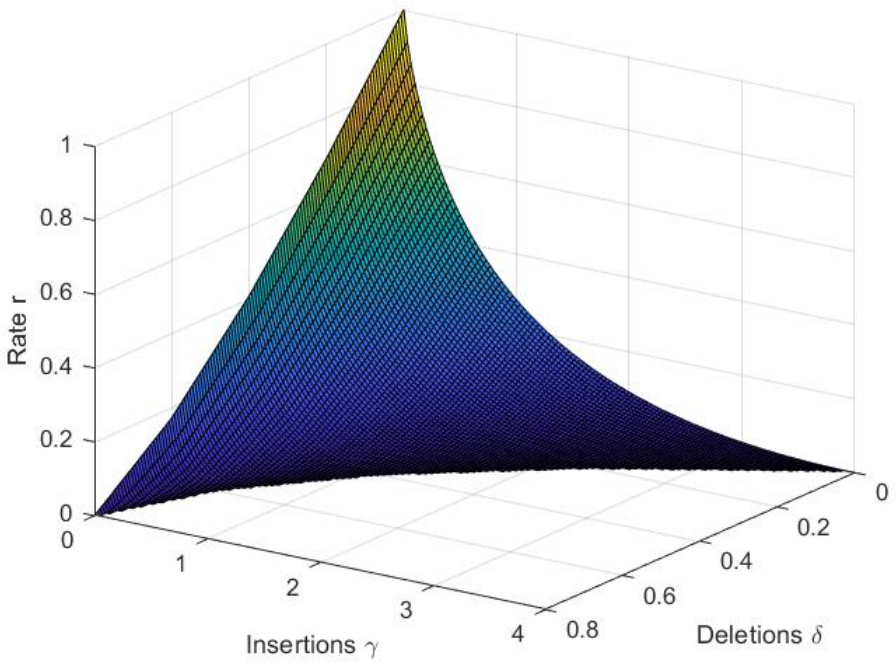


Figure 6.1: Depiction of the rate upper-bound from Theorem 6.1.4 for $q = 5$.

Finally, we use the bounds presented above for the special cases of insertion-only and deletion only-codes to derive a bound on the rate of codes that correct from a combination of insertions and deletions in Theorem 6.1.4. For every fixed alphabet size q, this bound can be nicely plotted as a 3D-surface in a 3D-chart which plots the maximum communication rate on the z-axis for all γ and δ (plotted on the x- and y-axes respectively). See Fig. 6.1 for an example of such a 3D-plot.

Note that in this representation, the cut onto the xz-plane corresponds to an upper-bound in the special case of insertion-only setting that is addressed in Theorem 6.1.1. Similarly, a cut onto the yz-plane derives a result for the deletion-only case that is addressed in Theorem 6.1.2. Finally, the cut onto the yz-plane specifies for which error rate combinations of γ and δ the communication rate hits zero, i.e. the problem of error resilience which was fully answered in Chapter 5. See Fig. 6.2 for an illustration of these three cuts.

A notable property of the upper bound of Theorem 6.1.4 is that, for every q, it exactly matches the best previously known results on all three aforementioned projections/cuts. That is, the upper bound implied for deletion-only codes (i.e., the cut on $\gamma = 0$ plane) matches the deletion-only bound from Theorem 6.1.2. Similarly, it matched the insertion-only bound of Theorem 6.1.1 when restricted to the $\delta = 0$ plane. Finally, the error resilience implied by Theorem 6.1.4 (i.e., where the curve in Figure 6.2 hits the floor) precisely matches the list-decoding error resilience curve for insertions and deletions as identified by Chapter 5. As such, the bound in Theorem 6.1.4 fully encapsulates and truly generalizes the entirety of the current state-of-the-art of the fundamental rate-distance

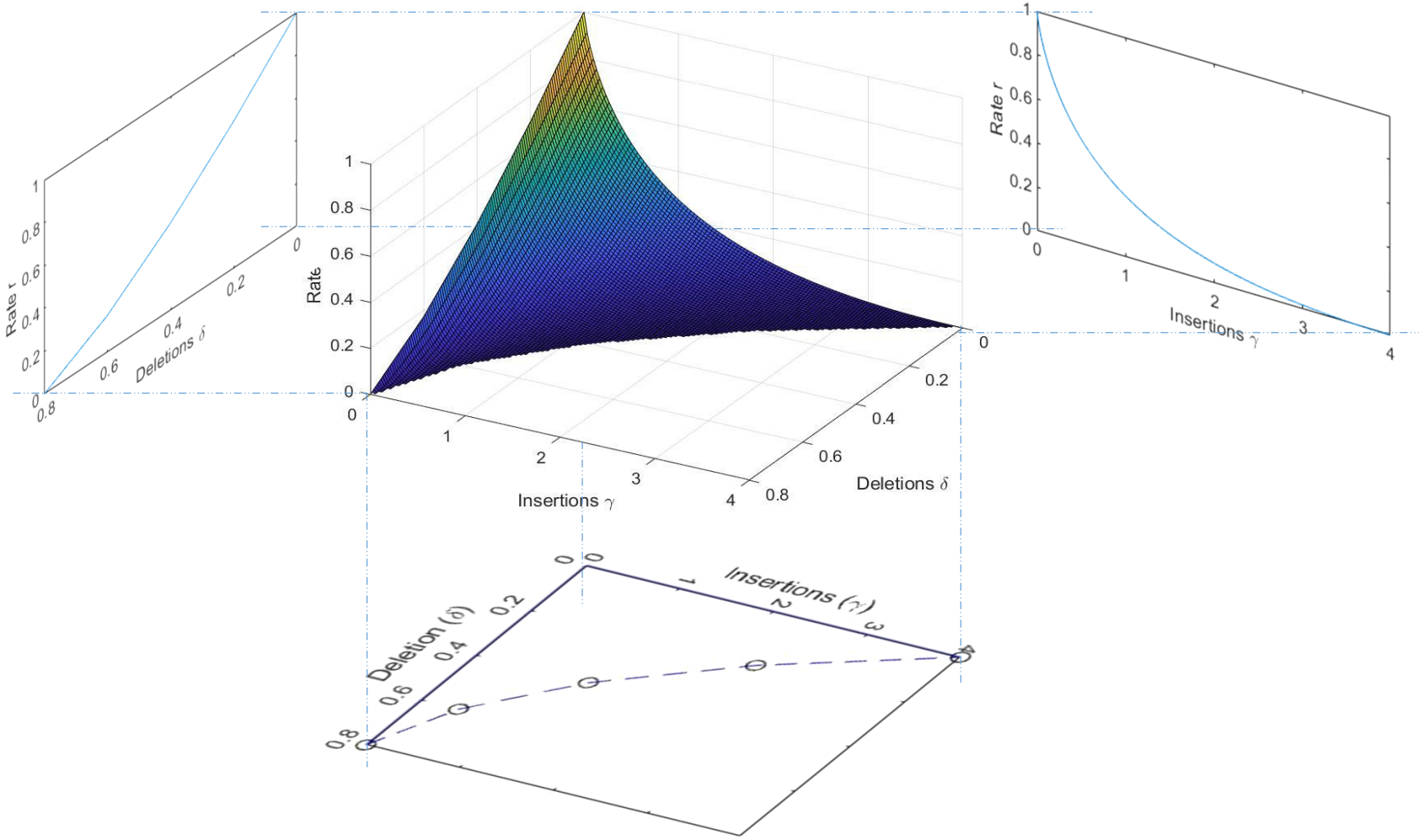


Figure 6.2: Depiction of our upper bound for $q = 5$ and its three projections for the insertion-only case (on the right), the deletion-only case (on the left), and the zero-rate case (on the bottom). The projection graphs are exactly matching the state-of-the-art results in the three special cases presented in Theorems 6.1.1 and 6.1.2 and Chapter 5.

trade-off for list-decodable insertion-deletion codes for any fixed alphabet size q.

Theorem 6.1.4. *Let C be a q-ary insertion-deletion code that is list-decodable from $\gamma \in [0, q-1]$ fraction of insertions and $\delta \in [0, 1-\frac{1}{q}]$. Then, the rate of C is no larger than*

$$\begin{aligned} & \alpha\left(1-\frac{d}{q}\right)\left((1+\gamma_0)\log_q\frac{q-d}{1+\gamma_0}-\gamma_0\log_q\frac{q-d-1}{\gamma_0}\right) \\ + \quad & (1-\alpha)\left(1-\frac{d+1}{q}\right)\left((1+\gamma_1)\log_q\frac{q-d-1}{1+\gamma_1}-\gamma_1\log_q\frac{q-d-2}{\gamma_1}\right) \end{aligned}$$

for $d = \lfloor \delta q \rfloor$, $\alpha = 1 - \delta q + d$, and all $\gamma_0, \gamma_1 \geq 0$ where $\alpha(1-\frac{d}{q})\gamma_0 + (1-\alpha)(1-\frac{d+1}{q})\gamma_1 = \gamma$. The optimal choice for γ_0 in this setting is

$$\gamma_0 = \frac{1}{2\alpha(q-d)} \cdot \left(A - \sqrt{B^2 + C}\right)$$

for

$$A = 3\alpha d^2 q + d^2 q - 3\alpha d q^2 - 2dq^2 + 4\alpha dq + 2dq + \alpha q^3 - 2\alpha q^2 + q^3 - 2q^2 + \gamma q + q - \alpha d^3 - 2\alpha d^2$$

$$B = \alpha d^3 + 2\alpha d^2 - 3\alpha d^2 q - d^2 q + 3\alpha d q^2 + 2dq^2 - 4\alpha dq - 2dq - \alpha q^3 + 2\alpha q^2 - q^3 + 2q^2 - \gamma q - q$$

$$C = 4(\alpha q - \alpha d)\left(\gamma d^2 q - 2\gamma dq^2 + 2\gamma dq + \gamma q^3 - 2\gamma q^2\right) \tag{6.1}$$

Fig. 6.1 depicts this upper-bound for $q = 5$.

6.1.2 Implications for Unique Decoding

Even though the main thrust of this chapter is list-decoding, Corollary 6.1.3 also has implications for unique-decoding. (This turns out to be a consequence of the fact that the list-decoding radius for deletions-only equals the unique-decoding radius for the same fraction of deletions.) We start by recalling the main result of Chapter 3: Given any $\alpha, \varepsilon > 0$ there exists a code of rate $1 - \alpha - \varepsilon$ over an alphabet of size $q = \exp(1/\varepsilon)$ that *uniquely* decodes from any α-fraction synchronization errors, i.e., from γ-fraction insertions and δ-fraction deletions for any pair $0 \leq \gamma, \delta$ satisfying $\gamma + \delta \leq \alpha$. Furthermore, this is the best possible rate one can achieve for α-fraction synchronization error. (See Section 6.5 for a more detailed description with proof.)

Prior to this work, this exponential dependence of the alphabet size on ε was unexplained. This is also in sharp contrast to the Hamming error setting, where codes are known to get ε close to unique decoding capacity (half the "Singleton bound" on the distance of code) with alphabets of size polynomial in $1/\varepsilon$. Indeed, given this contrast one may be tempted to believe that the exponential growth is a weakness of the "indexing-based synchronization string" approach presented in Chapter 3. But Corollary 6.1.3 actually shows that an exponential bound is necessary. We state this result for completeness even though it is immediate from the Corollary above, to stress its importance in understanding the nature of synchronization errors.

Corollary 6.1.5. *There exists a function $f : (0,1) \to (0,1)$ such that for every $\alpha, \varepsilon > 0$ every family of insdel codes of rate $1-\alpha-\varepsilon$ that protects against α-fraction of synchronization errors with unique decoding must have alphabet size $q \geq \exp\left(\frac{f(\delta)}{\varepsilon}\right)$.*

Corollary 6.1.5 follows immediately from Corollary 6.1.3 by setting $\delta = \alpha$ and $\gamma = 0$ (so we get to the zero insertion case) and noticing that a unique-decoding insdel code for α-fraction synchronization error is also a list-decoding insdel code for δ-fractions of deletions (and no insertions). The alphabet size lower-bound for the latter is also an alphabet size lower-bound for the former.

6.1.3 Lower-Bounds on Rate: Analysis of Random Codes

Finally, in Section 6.4, we provide an analysis of random codes and compute the rates they can achieve while maintaining list-decodability against insertions and deletions. Such rates are essentially lower-bounds for the capacity of insertion and deletion channels and can be compared against the upper-bounds provided in Section 6.2.

Theorem 6.1.6 shows that the family of random codes over an alphabet of size q can, with high probability, protect against δ-fraction of deletions for any $\delta < 1 - 1/q$ up to a rate of $1 - (1-\delta)\log_q \frac{1}{1-\delta} - \delta \log_q \frac{1}{\delta} - \delta \log_q(q-1) = 1 - H_q(\delta)$ using list decoding with super-constant list sizes in terms of their block length where H_q represents the q-ary entropy function.

Theorem 6.1.6. *For any alphabet of size q and any $0 \leq \delta < \frac{q-1}{q}$, the family of random codes with rate $R < 1 - (1-\delta)\log_q \frac{1}{1-\delta} - \delta \log_q \frac{1}{\delta} - \delta \log_q(q-1) - \frac{1-\delta}{l+1}$ is list-decodable with list size of l from any δ fraction of deletions with high probability. Further, the family of random deletion-codes with rate $R > 1 - (1-\delta)\log_q \frac{1}{1-\delta} - \delta \log_q \frac{1}{\delta} - \delta \log_q(q-1)$ is not list-decodable with high probability.*

Further, Theorem 6.1.7 shows that the family of random block codes over an alphabet of size q can, with high probability, protect against γ fraction of insertions for any $\gamma < q-1$ up to a rate of $1 - \log_q(\gamma+1) - \gamma \log_q \frac{\gamma+1}{\gamma}$ using list decoding with super-constant list sizes in terms of block length.

Theorem 6.1.7. *For any alphabet of size q and any $\gamma < q-1$, the family of random codes with rate $R < 1 - \log_q(\gamma+1) - \gamma \log_q \frac{\gamma+1}{\gamma} - \frac{\gamma+1}{l+1}$ is list-decodable with a list size of l from any γn insertions with high probability.*

Similar to the upper-bound results, we provide the following theorem that gives a lower-bound on the maximum achievable rate of list-decodable insertion-deletion codes. This result is also derived by analysis of the list-decodability of random codes.

Theorem 6.1.8. *For any integer $q \geq 2$ and $\delta \in \left[0, 1 - \frac{1}{q}\right]$ and $\gamma \in [0, q-1]$, a family of random codes with rate*

$$R < 1 - (1 - \delta + \gamma) H_q\left(\frac{\gamma}{1-\delta+\gamma}\right) - H_q(\delta) + \gamma \log_q(q-1)$$

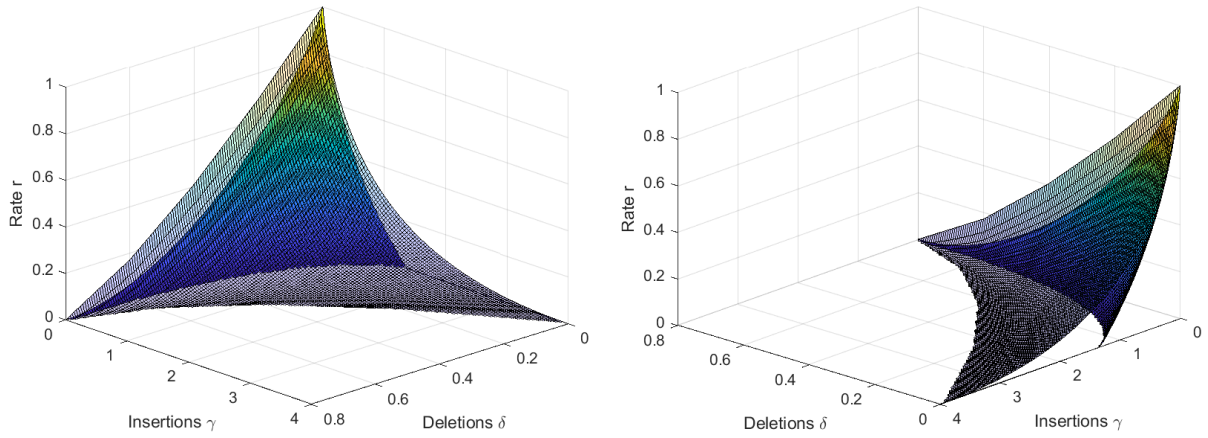


Figure 6.3: Depiction of our lower and upper bounds for $q = 5$ from two angles. The more transparent surface is the upper bound of Theorem 6.1.4 and the surface underneath is the lower bound derived in Theorem 6.1.8.

is list-decodable from a fraction γ of insertions and a fraction δ of deletions with high probability.

Fig. 6.3 illustrates the lower-bound implied by Theorem 6.1.8 in contrast to the upper-bound depicted in Fig. 6.1 for $q = 5$.

6.1.4 Organization of the Chapter

In Section 6.2, we proved proof for the rate upper-bounds presented in Section 6.1.1. We provide the proof of Corollary 6.1.3 in Section 6.3. We give the details of our random code analysis as summarized in Section 6.1.3 in Section 6.4 and, finally, present a formal argument for a claim made in Section 6.1.2 in Section 6.5 for the sake of completeness.

6.2 Upper Bounds on Rate

6.2.1 Deletion Codes (Theorem 6.1.2)

Proof of Theorem 6.1.2. To prove this claim, we propose a strategy for the adversary which can reduce the number of strings that may possibly arrive at the receiving side to a number small enough that implies the claimed upper bound for the rate.

We start by proving the theorem for the case where δq is integer. For an arbitrary code $\mathcal{C}$, upon transmission of any codeword, the adversary can remove all occurrences of δq least frequent symbols as the total number of appearances of such symbols does not exceed δn. In case there are more deletions left, adversary may choose to remove arbitrary symbols among the remaining ones. This way, the received string would be a string of $n(1-\delta)$ symbols consisted of only $q - q\delta$ many distinct symbols. Therefore, one can bound above

the size of the ensemble of strings that can possibly be received by the following.

$$|\mathcal{E}| \leq \binom{q}{q(1-\delta)} [q(1-\delta)]^{n(1-\delta)}$$

As the best rate that any $L = \text{poly}(n)$-list decodable code can get is at most $\frac{\log(|\mathcal{E}| \cdot L)}{n \log q} = \frac{\log |\mathcal{E}|}{n \log q} + o(1)$, the following would be an upper bound for the best rate one might hope for.

$$\begin{aligned} R &\leq \frac{\log |\mathcal{E}|}{n \log q} + o(1) \\ &= \frac{\log \binom{q}{q(1-\delta)} + n(1-\delta)(\log(q(1-\delta)))}{n \log q} + o(1) \\ &= (1-\delta)\left(1 - \log_q \frac{1}{1-\delta}\right) + o(1) \end{aligned}$$

This shows that for the case where $q\delta$ is an integer number, there are no family of codes that achieve a rate that is strictly larger than $(1-\delta)\left(1 - \log_q \frac{1}{1-\delta}\right)$.

We now proceed to the general case where $\delta = d/q + \delta'$ for some integer d and $0 \leq \delta' < \frac{1}{q}$. We closely follow the idea that we utilized for the former case. The adversary can partition n sent symbols into two parts of size $nq\delta'$ and $n(1-q\delta')$, and then, similar to the former case, removes the $d+1$ least frequent symbols from the first part by performing $\frac{d+1}{q} \cdot nq\delta'$ deletions and d least frequent symbols from the second one by performing $\frac{d}{q} \cdot n(1-q\delta')$ ones. This is possible because $\frac{d+1}{q} \cdot nq\delta' + \frac{d}{q} \cdot n(1-q\delta') = n\delta$. Doing so, the string received after deletions would contain up to $q-d-1$ distinct symbols in its first $nq\delta'(1-(d+1)/q)$ positions and up to $q-d$ distinct symbols in the other $n(1-q\delta')(1-d/q)$ positions. Therefore, the size of the ensemble of strings that can be received is bounded above as follows.

$$|\mathcal{E}| \leq \binom{q}{q-d-1} [q-d-1]^{nq\delta'\left(1-\frac{d+1}{q}\right)} \cdot \binom{q}{q-d} [q-d]^{n(1-q\delta')\left(1-\frac{d}{q}\right)}$$

This bounds above the rate of any family of list-decodable insdel codes by the following.

$$\begin{aligned} &\frac{\log |\mathcal{E}|}{n \log q} \\ &= \frac{\log \binom{q}{q-d-1} + nq\delta'\left(1-\frac{d+1}{q}\right)\log(q-d-1) + \log \binom{q}{q-d} + n(1-q\delta')\left(1-\frac{d}{q}\right)\log(q-d)}{n \log q} \\ &= q\delta'\left[\left(1-\frac{d+1}{q}\right)\left(1-\log_q \frac{1}{1-(d+1)/q}\right)\right] + (1-q\delta')\left[\left(1-\frac{d}{q}\right)\left(1-\log_q \frac{1}{1-d/q}\right)\right] \end{aligned}$$

□

6.2.2 Insertion Codes (Theorem 6.1.1)

Before providing the proof of Theorem 6.1.1, we first point out that any $q-1$ insertions can be essentially used as a single erasure. As a matter of fact, by inserting $q-1$ symbols around the first symbol adversary can make a $1, 2, \cdots, q$ substring around first symbol and therefore, essentially, make the receiver unable to gain any information about it. In fact, with γn insertions, the adversary can repeat this procedure around any $\left\lfloor \frac{\gamma n}{q-1} \right\rfloor$ symbols he wishes. This basically gives that, with γn insertions, adversary can *erase* $\left\lfloor \frac{\gamma n}{q-1} \right\rfloor$ many symbols. Thus, one cannot hope for finding list-decodable codes with rate $1 - \frac{\gamma}{q-1}$ or more protecting against γn insertions.

Proof of Theorem 6.1.1. To prove this, consider a code $\mathcal{C}$ with rate $R \geq 1 - \log_q(\gamma + 1) - \gamma \left(\log_q \frac{\gamma+1}{\gamma} - \log_q \frac{q}{q-1} \right) + \varepsilon$ for some $\varepsilon > 0$. We will show that there exist c_0^n many codewords in C that can be turned into one specific string $z \in [1..q]^{n(\gamma+1)}$ with γn insertions for some constant $c_0 > 1$ that merely depends on q and ε.

First, the lower bound assumed for the rate implies that

$$|C| = q^{nR} \geq q^{n\left(1 - \log_q(\gamma+1) - \gamma\left(\log_q \frac{\gamma+1}{\gamma} - \log_q \frac{q}{q-1}\right) + \varepsilon\right)}. \tag{6.2}$$

Let Z be a random string of length $(\gamma+1)n$ over the alphabet $[1..q]$. We compute the expected number of codewords of $\mathcal{C}$ that are subsequences of Z denoted by X.

$$\begin{aligned}
\mathbb{E}[X] &= \sum_{y \in \mathcal{C}} \Pr\{y \text{ is a subsequence of } Z\} \\
&= \sum_{y \in \mathcal{C}} \sum_{1 \leq a_1 < a_2 < \cdots < a_n \leq n(\gamma+1)} \frac{1}{q^n} \left(1 - \frac{1}{q}\right)^{a_n - n} && (6.3) \\
&= |\mathcal{C}| \, (q-1)^{-n} \sum_{l=n}^{n(1+\gamma)} \binom{l}{n} \left(\frac{q-1}{q}\right)^l \\
&\geq |\mathcal{C}| \, (q-1)^{-n} \binom{n(1+\gamma)}{n} \left(\frac{q-1}{q}\right)^{n(1+\gamma)} && (6.4) \\
&= n\gamma |\mathcal{C}| (q-1)^{n\gamma} q^{-n(1+\gamma)} 2^{n(1+\gamma) H\left(\frac{1}{1+\gamma}\right) + o(n)} \\
&= n\gamma |\mathcal{C}| q^{n\left(\gamma \log_q(q-1) - 1 - \gamma + \log_q(1+\gamma) + \gamma \log_q \frac{1+\gamma}{\gamma}\right) + o(1)} \\
&= q^{n\varepsilon + o(n)} && (6.5)
\end{aligned}$$

Step (6.6) is obtained by conditioning the probability of y being a subsequence of Z over the leftmost occurrence of y in Z indicated by $a_1, a_2, \cdots, a_n$ as indices of Z where the leftmost occurrence of y is located. In that event, Z_{a_i} has to be similar to y_i and y_i cannot appear in $Z[a_{i-1}+1, a_i - 1]$. Therefore, the probability of this event is $\left(\frac{1}{q}\right)^n \left(1 - \frac{1}{q}\right)^{a_n - n}$.

By (6.8), there exists some $z \in [1..q]^{(a+1)n}$ to which at least $q^{\varepsilon n + o(n)}$, i.e., exponentially many codewords of $\mathcal{C}$ are subsequences. Therefore, polynomial-sized list decoding for received message z is impossible and proof is complete.

We would like to remark that the lower-bound in Step (6.7) is asymptotically tight. To verify this, we show that the summation in previous step takes its largest value when $l = n(1+\gamma)$ and bound the summation above by $n\gamma$ times that term. To see that $\binom{l}{n}\left(\frac{q-1}{q}\right)^l$ is maximized for $l = n(1+\gamma)$ in $n \leq l \leq n(1+\gamma)$ it suffices to show that the ratio of consecutive terms is larger than one for $l \leq n(1+\gamma)$:

$$\frac{\binom{l}{n}\left(\frac{q-1}{q}\right)^l}{\binom{l-1}{n}\left(\frac{q-1}{q}\right)^{l-1}} = \frac{l}{l-n} \cdot \frac{q-1}{q} = \frac{1-\frac{1}{q}}{1-\frac{n}{l}} \geq 1$$

The last inequality follows from the fact that $l \leq n(\gamma+1) \leq nq \Rightarrow \frac{1}{q} < \frac{n}{l}$.

□

6.2.3 Insertion-Deletion Codes (Theorem 6.1.4)

Linear Upper Bounds using Resilience Results

Before discussing Theorem 6.1.4, we present a simple upper-bound using the error-resilience region identified in Theorem 5.1.3. We show that the multiplicative distance to this region gives a valid upper-bound on the rate of any list-decodable insertion-deletion code:

Theorem 6.2.1. *For any alphabet size q and any $(\gamma, \delta) \in F_q$ let $\alpha \geq 1$ be the smallest number such that $(\alpha\gamma, \alpha\delta) \notin R_q$. Any family of (γ, δ)-list decodable q-ary codes cannot achieve a rate of more than $1 - 1/\alpha$.*

A different way to look at this upper bound is to think of it as the collection of lines that connect every point on the infeasibility region F_q identified in Theorem 5.1.3 on $c = 0$ plane and the point $(\gamma, \delta, c) = (0, 0, 1)$ which indicates the trivial achievable rate of 1 in the absence of noise. (See Fig. 6.4)

The proof of the theorem above is easy once one recalls how the upper bound for the feasibility region R_q is proven in Chapter 5. It basically consists of a simple strategy transforming any sent string into one of a small $O_q(1)$ number of canonical strings, thus erasing almost all information sent. One can prove Theorem 6.2.1 by doing the same but only on an $\frac{1}{\alpha}$ fraction of the string. We present an alternative and shorter formal proof in the following.

Proof of Theorem 6.2.1. Assume by contradiction that family of (γ, δ)-list-decodable codes $\mathcal{C} = \{C_1, C_2, \cdots\}$ with block lengths $n_1 < n_2 < \cdots$ and rates $r_1, r_2, \cdots$ achieve a rate of $r = \lim_{i\to\infty} r_i = 1 - \frac{1}{\alpha} + \varepsilon$ for some $\varepsilon > 0$.

We convert this family of codes to a new family of codes $\mathcal{C}'$ by converting each code C_i into a code C_i' as follows: In all codewords of C_i, consider the $n_i\,(1 - 1/\alpha)$-long prefix. Among all such prefixes, let p be the most frequent one. We set C_i' to be a code containing

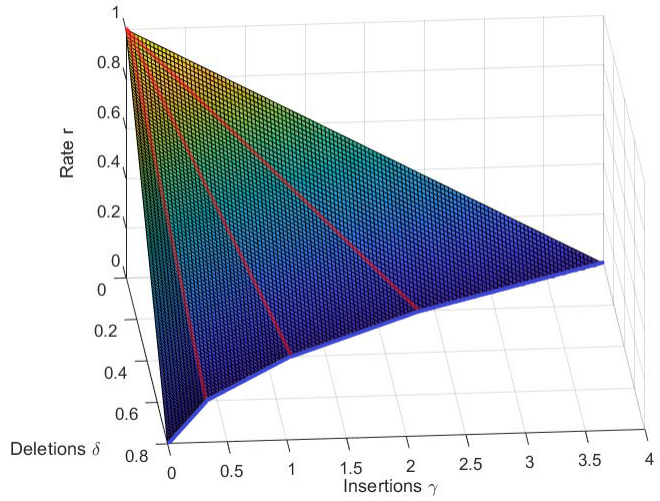


Figure 6.4: Illustration of the upper bound from Theorem 6.2.1 for $q = 5$.

all codewords of C_i that start with p. Since all such codewords start with p, we omit the prefix p from all such codewords. Note that the block length of C_i' is $n_i' = n_i - n_i\,(1 - 1/\alpha) = n_i/\alpha$. Also, since there are $q^{n_i(1-1/\alpha)}$ q-ary strings of length $n_i\,(1 - 1/\alpha)$,

$$|C_i'| \geq \frac{|C_i|}{q^{n_i(1-1/\alpha)}} = \frac{q^{n_i r_i}}{q^{n_i(1-1/\alpha)}} = q^{n_i(r_i - 1 + 1/\alpha)}.$$

This implies that the rate of C_i' is at least

$$r_i' = \frac{\log_q |C_i'|}{n_i'} \geq \frac{n_i(r_i - 1 + 1/\alpha)}{n_i'} = \alpha(r_i - 1 + 1/\alpha)$$

and, hence, the rate of the family of codes $\mathcal{C}'$ is at least $\lim_{i\to\infty} r_i' \geq \alpha\varepsilon > 0$.

Further, we claim that if $\mathcal{C}$ is $(\gamma, \delta, L(n))$-list decodable, then $\mathcal{C}'$ will be $(\alpha\gamma, \alpha\delta, L(\alpha n'))$-list decodable. To show this, we construct such list-decoder for all codes $C_i' \in \mathcal{C}'$ with input y' by simply padding the most frequent $n_i(1 - 1/\alpha)$-prefix of codewords of $C_i \in \mathcal{C}$, p, in front of y' and running the list-decoder of C_i with input $y = p \cdot y'$. Among the list generated by the decoder of C_i, the ones that do not start with p are withdrawn. The remaining strings will form the output of our list-decoder for C_i' after omitting their prefix p. Note that this indeed gives a $(\alpha\gamma, \alpha\delta, L(\alpha n'))$-list-decoder since for any codeword $x \in C_i'$ that is $(\alpha\gamma, \alpha\delta)$-close to y, $p \cdot x \in C_i$ is (γ, δ)-close to $p \cdot y$.

Note that we were able to show that the family of codes $\mathcal{C}'$ achieves a positive rate and is $(\alpha\gamma, \alpha\delta)$-list decodable for $(\alpha\gamma, \alpha\delta) \notin F_q$. This is a contradiction to Theorem 5.1.3 proving that the rate of $\mathcal{C}'$ may not exceed $1 - \frac{1}{\alpha}$, thus, proving the theorem. □

Upper Bounds and Convexity of the Unachievability Region.

We would like to remark that one can interpret the upper bound from Theorem 6.2.1 in the following manner: We know that no code can achieve a rate larger than one even in

the absence of noise, i.e., the region $R_1 = \{(\gamma, \delta, r) \mid \gamma = \delta = 0, r > 1\}$ is "infeasible". Further, the resilience result from Chapter 5 demonstrates that all points within $R_2 = \{(\gamma, \delta, 0) \mid (\gamma, \delta) \notin F_q\}$ are infeasible. Theorem 6.2.1 shows that the convex combination of R_1 and R_2 is infeasible as well, implying a pyramid-shaped feasibility region with F_q as its base and $(0, 0, 1)$ as its apex. (See Figure 6.4.) Also, note that the argument in the proof of Theorem 6.2.1 is generic and independent of the shape of F_q. This is similar to some known upper bounds for ordinary error-correcting codes which essentially present a convex generalization between two known infeasible points such as the Singleton bound. While being more complicated and less generic, our tighter upper bound in Theorem 6.1.4 also fits the same pattern of showing infeasibility of convex combinations of known infeasible points. It is, however, unknown to us whether one can generically prove the convexity of the infeasibility region in this setting.

Correctness of Theorem 6.1.4

We now proceed to discussing Theorem 6.1.4. Before providing the proof of Theorem 6.1.4, we give a simple representation of the bound set forth in Theorem 6.1.4. Consider a code C that is (γ, δ)-list-decodable and assume that $\delta = \frac{d}{q}$ for some integer q. We propose a strategy for the adversary as follows: The adversary can use the $\frac{d}{q}$ deletion to remove all occurrences of the d-least frequent symbols of the alphabet and hence transform the codewords to strings of length $n(1-\delta)$ over an alphabet of size $q(1-\delta) = q - d$. We can then apply the bound that we have for insertion-only codes to derive an upper bound on the size and, thus, the rate of the code. For the general case of δ not necessarily being an integer multiply of $\frac{1}{q}$, we use a time-sharing argument to show that the points obtained by the linear combination of upper-bound in this special case serve as upper-bounds for the rest of (γ, δ) pairs. See Fig. 6.5 for a representation of this bound for $q = 5$ (both the special case of $\delta = \frac{d}{q}$ and the general case are depicted).

We start with the statement of bound for the special case of $\delta = \frac{d}{q}$.

Theorem 6.2.2. *For any alphabet size q, any insertion rate $\gamma < q - 1$ and any deletion rate $\delta = \frac{d}{q}$ for some integer $d \leq q$, it is true that any family of q-ary codes $\mathcal{C}$ which is (γ, δ)-list-decodable has a rate of at most* $(1-\delta)\left[\left(1 + \frac{\gamma}{1-\delta}\right)\log_q \frac{q-d}{\frac{\gamma}{1-\delta}+1} - \frac{\gamma}{1-\delta} \cdot \left(\log_q \frac{q-d-1}{\frac{\gamma}{1-\delta}}\right)\right]$.

Proof. Consider a code C that is (γ, δ)-list-decodable and assume that $\delta = \frac{d}{q}$ for some integer d. Assume that we restrict the adversary to utilize its deletions in the following manner: The adversary uses the $\frac{d}{q}$ deletion to remove all occurrences of the d-least frequent symbols of the alphabet. If there are remaining deletions, the adversary removes symbols from the end of the transmitted word.

Let us define the code C' that is obtained from C by deleting a δ fraction of symbols from each codeword of C as described above. Note that the block length of C' is $n' = n(1-\delta)$ and each of its codewords consist of up to $q' = q(1-\delta) = q - d$ symbols of the alphabet though this subset of size $q-d$ may be different from codeword to codeword. We partition the codewords of C' into $\binom{q}{q-d}$ sets $C'_1, C'_2, \cdots, C'_{\binom{q}{q-d}}$ based on which $(q-d)$-subset of the alphabet they consist of.

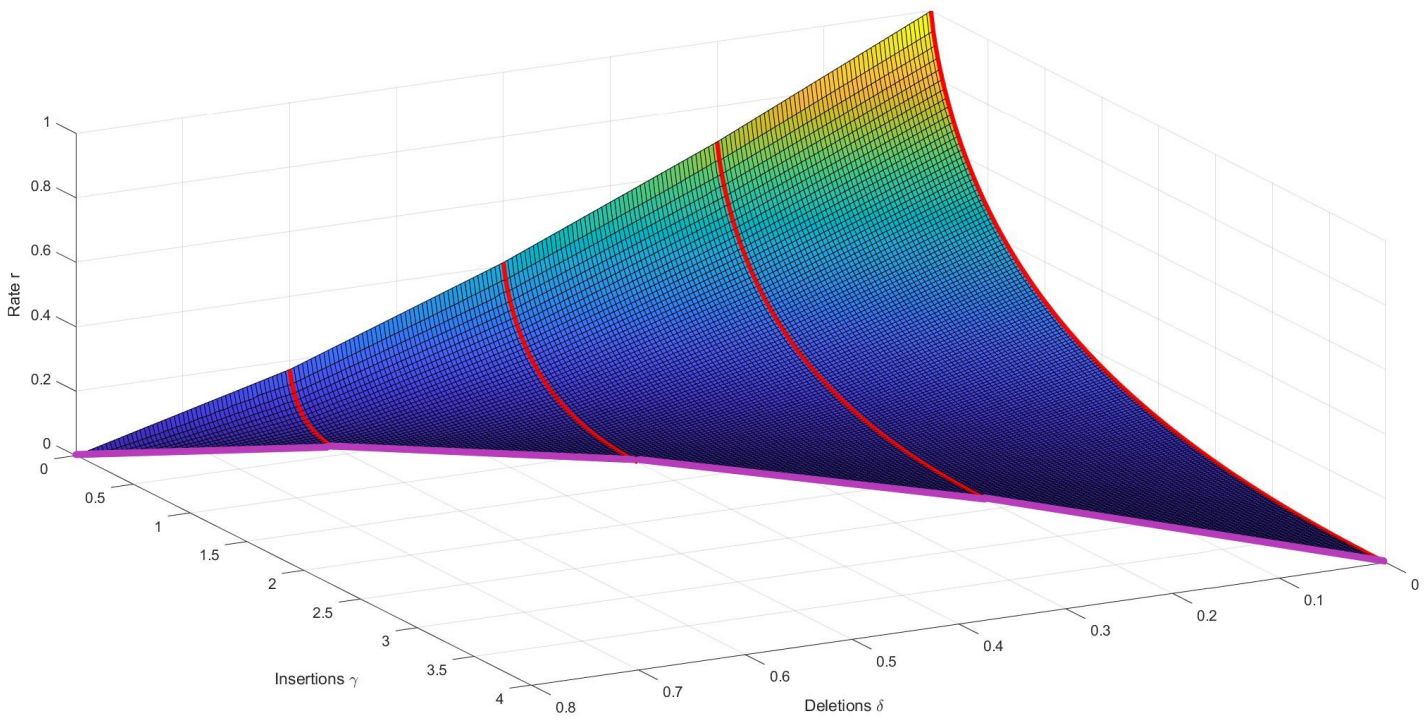


Figure 6.5: Upper bound for rate for $q = 5$. The special case where $\delta = \frac{d}{q}$ for some integer d is indicated with red lines.

Since C is (γ, δ)-list-decodable, each of the C_i's are list-decodable from γn insertions. Therefore, Theorem 6.1.1 implies that the size of each code C_i' is no larger than:

$$q'^{n'\left[1-\log_{q'}(\gamma'+1)-\gamma'\left(\log_{q'}\frac{\gamma'+1}{\gamma'}-\log_{q'}\frac{q'}{q'-1}\right)\right]}$$

where $q' = q - d$, $n' = n(1-\delta)$, and $\gamma' = \frac{\gamma}{1-\delta}$. Therefore, the size of the code C is no larger than

$$\binom{q}{q-d} q^{n(1-\delta)\left[\log_q q' - \log_q(\gamma'+1) - \gamma'\left(\log_q \frac{\gamma'+1}{\gamma'} - \log_q \frac{q'}{q'-1}\right)\right]}$$

and, consequently, its rate is no larger than

$$\begin{aligned} & (1-\delta)\left[\log_q(q-d) - \log_q\left(\tfrac{\gamma}{1-\delta}+1\right) - \tfrac{\gamma}{1-\delta}\cdot\left(\log_q\tfrac{\gamma+1-\delta}{\gamma} - \log_q\tfrac{q-d}{q-d-1}\right)\right] \\ = \quad & (1-\delta)\left[\left(1+\tfrac{\gamma}{1-\delta}\right)\log_q\tfrac{q-d}{\frac{\gamma}{1-\delta}+1} - \tfrac{\gamma}{1-\delta}\cdot\left(\log_q\tfrac{q-d-1}{\frac{\gamma}{1-\delta}}\right)\right]. \end{aligned}$$

□

Given the nice and explicit form of Theorem 6.2.2 for any q and γ with multiple specific values of δ, it seems tempting to conjecture that the restriction of δ is unnecessary making $(1-\delta)\left[\left(1+\frac{\gamma}{1-\delta}\right)\log_q\frac{q-d}{\frac{\gamma}{1-\delta}+1} - \frac{\gamma}{1-\delta}\cdot\left(\log_q\frac{q-d-1}{\frac{\gamma}{1-\delta}}\right)\right]$ a valid upper bound for any value of δ (and γ). This, however, could not be further from the truth. Indeed, for any δ not of the form restricted to by Theorem 6.2.2, there exists a γ for which this extended bound is provably wrong because it contradicts the existence of certain list-decodable codes constructed in Chapter 5.

In fact, for the valid points where δ is a multiple of $\frac{1}{q}$, the rate bound of Theorem 6.2.2 hits zero at exactly the corner points of the piece-wise linear resilience region F_q characterized by Chapter 5. Taking this as an inspiration, one could try to extend the bound of Theorem 6.2.2 to all values of δ by considering for each q and each rate r the roughly $\frac{q}{r}$ points where Theorem 6.2.2 hits the plane corresponding to rate r and extend these points in a piece-wise linear manner to a complete 2D-curve for this rate r. This would give a rate bound for any γ, δ, and q as desired, which reduces to a piece-wise linear function for any fixed r and also correctly reproduce F_q for $r = 0$.

It turns out that this is indeed a correct upper bound. However, a stronger form of convexity, which takes full 3D-convex interpolations between any points supplied by Theorem 6.2.2 and in particular combines points with different rates, also holds and is needed to give our final upper bound.

Theorem 6.2.3. *For a fixed q suppose that $(\gamma_0, \delta_0 = \frac{d_0}{q})$ and $(\gamma_1, \delta_1 = \frac{d_1}{q})$ are two error rate combinations for which Theorem 6.2.2 implies a maximal communication rate of r_0 and r_1, respectively. For any $0 \le \alpha \le 1$ consider the following convex combinations of these quantities: $\gamma = \alpha\gamma_0 + (1-\alpha)\gamma_1$, $\delta = \alpha\delta_0 + (1-\alpha)\delta_1$, and $r = \alpha r_0 + (1-\alpha)r_1$. It is true that any (δ, γ)-list-decodable q-ary code has a rate of at most r.*

See Figure 6.1 for an illustration of this bound for $q = 5$. Red curves indicate the upper bound described above for the special values of δ of the form $\frac{d}{q}$ as given by Theorem 6.2.2.

Theorem 6.2.3 together with Theorem 6.2.2 gives a conceptually very clean description of our upper bound. However, an (exact) evaluation of the upper bound as given by Theorem 6.2.3 is not straightforward since there are many convex combinations which all produce valid bound but how to compute or select the one which gives the strongest guarantee on the rate for a given (γ, δ) pair is not clear. This is particularly true since, as already mentioned above, the optimal points to combine do not lie on the same rate-plane.

Theorem 6.1.4 is an alternative statement to Theorem 6.2.3 which produces an explicit upper bound for any (γ, δ) as an α-convex combination of two points (γ_0, δ_0) and (γ_1, δ_1) only in dependence on the free parameter γ_0. It then shows an explicit expression for the optimal value for γ_0. This produces a significantly less clean but on the other hand fully explicit description of our upper bound.

Proof of Theorem 6.1.4 (and Theorem 6.1.4). We first note that the statements of Theorem 6.1.4 and Theorem 6.2.3 are merely a rephrasing of each other with the exception that Theorem 6.1.4 only allows and optimizes over convex combinations of neighboring spokes of Theorem 6.2.2, namely the ones for $d_0 = d$ and $d_1 = d + 1$ for $d = \lfloor \delta q \rfloor$. This restriction, however, is without loss of generality. Indeed, for any values from the domain $\left\{(\gamma, \delta) \mid \delta = \frac{d}{q}, 0 \leq d \leq q-1, d \in \mathbb{Z}\right\}$, Theorem 6.2.2 gives values which come from the function $f(\gamma, \delta) = (1-\delta)\left[\left(1 + \frac{\gamma}{1-\delta}\right)\log_q \frac{q(1-\delta)}{\frac{\gamma}{1-\delta}+1} - \frac{\gamma}{1-\delta} \cdot \left(\log_q \frac{q(1-\delta)-1}{\frac{\gamma}{1-\delta}}\right)\right]$. This function $f(\gamma, \delta)$ is convex. (see Section 6.6 for a formal proof.) Any value given as a convex combination between two non-neighboring spokes can therefore be at least matched (and indeed, thanks to the strict convexity of $f(\cdot, \cdot)$ always be improved) by choosing a different convex combination between neighboring spokes. This justifies the "restricted" formulation of Theorem 6.1.4, which helps in reducing the number of parameters and simplifies calculations.

In order to prove Theorem 6.1.4 we, again, fix a specific strategy for the adversary's use of deletions. In particular, the adversary will use $n\alpha\frac{d}{q}$ deletions on the first $n\alpha$ symbols of the transmitted codeword to eliminate all instances of the d least-frequent symbols there. Similarly, he removes all instances of the respective $d+1$ least frequent symbols from the last $n(1-\alpha)$ symbols of the codeword. The resulting string is one out of some $\Sigma_0^{n\alpha(1-\frac{d}{q})} \times \Sigma_1^{n(1-\alpha)(1-\frac{d+1}{q})}$ where $\Sigma_0, \Sigma_1 \subseteq [q]$, $q_0 = |\Sigma_0| = q - d$, $q_1 = |\Sigma_1| = q - d - 1$. This deletion strategy fits within the budgeted number of deletions since $\delta = \alpha\frac{d}{q} + (1-\alpha)\frac{d+1}{q}$ for $d = \lfloor \delta q \rfloor$ and $\alpha = 1 - \delta q + d$.

Note that while the adversary can convert any codeword of C to a string of such form, the sub-alphabets Σ_0 and Σ_1 will likely be different between different codewords of C. Let (Σ_0, Σ_1) be the pair of the most frequently reduced to alphabets and let C_0 be the set of codewords of C that, after undergoing the above-described procedure, turn into a string out of $\Sigma_0^{n\alpha(1-\frac{d}{q})} \times \Sigma_1^{n(1-\alpha)(1-\frac{d+1}{q})}$. Note that $\frac{|C|}{\binom{q}{d}\binom{q}{d+1}} \leq |C_0| \leq |C|$. Further, let D_0 be the set of codewords in C_0 after undergoing the alphabet reduction procedure mentioned above. To give an upper bound of the rate of C it thus suffices to bound from above the size of C_0 (or equivalently, D_0.)

We bound above the size of D_0 by showing that if $|D_0|$ is too large, there will be some

received word that can be obtained by exponentially many words in D_0 after $n\gamma$ insertions. Similar to the insertion-only case, we utilize the linearity of expectation to derive this. Let us pick a random string $Z = (Z_0, Z_1)$ that consists of $n\alpha(1-\frac{d}{q})(1+\gamma_0)$ symbols chosen uniformly out of Σ_0 (referred to by Z_0) and $n(1-\alpha)(1-\frac{d+1}{q})(1+\gamma_1)$ symbols uniformly chosen out of Σ_1 (referred to by Z_1). We have that $\alpha(1-\frac{d}{q})\gamma_0 + (1-\alpha)(1-\frac{d+1}{q})\gamma_1 = \gamma$. ($\gamma_0$ and γ_1 will be determined later.) We calculate the expected number of the members of D_0 that are subsequences of such string – denoted by X. In the following, we will often describe members of D_0 like y as the concatenation (y_0, y_1) where $|y_0| = n_0 = n\alpha(1-\frac{d}{q})$ and $|y_1| = n_1 = n(1-\alpha)(1-\frac{d+1}{q})$.

$$
\begin{aligned}
\mathbb{E}[X] &= \sum_{y=(y_0,y_1)\in D_0} \Pr\{y \text{ is a subsequence of } Z\} \\
&\geq \sum_{y=(y_0,y_1)\in D_0} \Pr\{y_0 \text{ is a subsequence of } Z_0\} \cdot \Pr\{y_1 \text{ is a subsequence of } Z_1\} \\
&= \sum_{y=(y_0,y_1)\in D_0} \prod_{j=1,2} \Pr\{y_j \text{ is a subsequence of } Z_j\} \\
&= \sum_{y=(y_0,y_1)\in D_0} \prod_{j=1,2} \sum_{1\leq a_1<a_2<\cdots<a_{n_j}\leq n_j(1+\gamma_j)} \frac{1}{|\Sigma_j|^{n_j}} \left(1-\frac{1}{|\Sigma_j|}\right)^{a_{n_j}-n_j} \qquad (6.6) \\
&= |D_0| \prod_{j=1,2} (|\Sigma_j|-1)^{-n_j} \sum_{l=n_j}^{n_j(1+\gamma_j)} \binom{l}{n_j} \left(\frac{|\Sigma_j|-1}{|\Sigma_j|}\right)^{l} \\
&\geq |D_0| \prod_{j=1,2} (|\Sigma_j|-1)^{-n_j} \binom{n_j(1+\gamma_j)}{n_j} \left(\frac{|\Sigma_j|-1}{|\Sigma_j|}\right)^{n_j(1+\gamma_j)} \qquad (6.7) \\
&= |D_0| \prod_{j=1,2} (|\Sigma_j|-1)^{n_j\gamma_j} |\Sigma_j|^{-n_j(1+\gamma_j)} 2^{n_j(1+\gamma_j)H\left(\frac{1}{1+\gamma_j}\right)+o(n)} \\
&= |D_0| \prod_{j=1,2} q^{n_j\left(\gamma_j \log_q(q_j-1)-(1+\gamma_j)\log_q q_j+(1+\gamma_j)\log_q(1+\gamma_j)-\gamma_j\log_q\gamma_j\right)+o(1)} \\
&= |D_0| \prod_{j=0,1} q^{n_j\left(\gamma_j \log_q \frac{q_j-1}{\gamma_j}-(1+\gamma_j)\log_q\frac{q_j}{1+\gamma_j}\right)+o(1)} \\
&= |D_0| q^{\sum_{j=0,1} n_j\left(\gamma_j \log_q \frac{q_j-1}{\gamma_j}-(1+\gamma_j)\log_q\frac{q_j}{1+\gamma_j}\right)+o(1)} \qquad (6.8)
\end{aligned}
$$

Step (6.6) is obtained by conditioning the probability of y_j being a subsequence of Z_j over the leftmost occurrence of y_j in Z_j indicated by $a_1, a_2, \cdots, a_n$ as indices of Z_j where the leftmost occurrence of y_j is located. In that event, $Z_j[a_i]$ has to be similar to $y_j[i]$ and $y_j[i]$ cannot appear in $Z_j[a_{i-1}+1, a_i-1]$. Therefore, the probability of this event is $\left(\frac{1}{q_j}\right)^n_j \left(1-\frac{1}{q_j}\right)^{a_n-n_j}$. To verify Step (6.7), we show that the summation in previous step takes its largest value when $l = n_j(1+\gamma_j)$ and bound the summation below by that term.

To see that $\binom{l}{n_j}\left(\frac{q_j-1}{q_j}\right)^l$ is maximized for $l=n_j(1+\gamma_j)$ in $n_j \le l \le n_j(1+\gamma_j)$, it suffices to show that the ratio of consecutive terms is larger than one for $l \le n_j(1+\gamma_j)$:

$$\frac{\binom{l}{n_j}\left(\frac{q_j-1}{q_j}\right)^l}{\binom{l-1}{n_j}\left(\frac{q_j-1}{q_j}\right)^{l-1}} = \frac{l}{l-n_j}\cdot\frac{q_j-1}{q_j} = \frac{1-\frac{1}{q_j}}{1-\frac{n_j}{l}} \ge 1$$

The last inequality follows from the fact that $l \le n_j(\gamma_j+1) \le n_j q_j \Rightarrow \frac{1}{q_j} < \frac{n_j}{l}$.

Finally, by (6.8), there exists some realization of Z to which at least

$$|D_0| q^{\sum_{j=0,1} n_j\left(\gamma_j \log_q \frac{q_j-1}{\gamma_j} - (1+\gamma_j)\log_q \frac{q_j}{1+\gamma_j}\right)+o(1)}$$

codewords of $\mathcal{C}$ are subsequences. In order for C to be list-decodable, this quantity needs to be sub-exponential. Therefore,

$$\begin{aligned}
r_C &= \frac{\log_q |D_0| + O(1)}{n} \le \sum_{j=0,1}\frac{n_j}{n}\left((1+\gamma_j)\log_q\frac{q_j}{1+\gamma_j} - \gamma_j\log_q\frac{q_j-1}{\gamma_j}\right)\\
&= \alpha\left(1-\frac{d}{q}\right)\left((1+\gamma_0)\log_q\frac{q_0}{1+\gamma_0} - \gamma_0\log_q\frac{q_0-1}{\gamma_0}\right)\\
&\quad +(1-\alpha)\left(1-\frac{d+1}{q}\right)\left((1+\gamma_1)\log_q\frac{q_1}{1+\gamma_1} - \gamma_1\log_q\frac{q_1-1}{\gamma_1}\right)\\
&= \alpha\left(1-\frac{d}{q}\right)\left((1+\gamma_0)\log_q\frac{q-d}{1+\gamma_0} - \gamma_0\log_q\frac{q-d-1}{\gamma_0}\right)\\
&\quad +(1-\alpha)\left(1-\frac{d+1}{q}\right)\left((1+\gamma_1)\log_q\frac{q-d-1}{1+\gamma_1} - \gamma_1\log_q\frac{q-d-2}{\gamma_1}\right) \qquad (6.9)
\end{aligned}$$

Note that (6.9) is an upper bound for the rate for all choices of $\gamma_0, \gamma_1 \ge 0$ where $\alpha(1-\frac{d}{q})\gamma_0 + (1-\alpha)(1-\frac{d+1}{q})\gamma_1 = \gamma$.

To find the optimal value for γ_0, we find the choice of γ_0 that minimizes (6.6). To this end, we calculate the ratio between the values of (6.6) when $n\gamma$ insertions are distributed between two parts as $(n_0\gamma_0, n_1\gamma_1)$ and when distributed as $(n_0\gamma_0+1, n_1\gamma_1-1)$. We then find out the choice of γ_0 for which this ratio exceeds one. This would indicate the value of γ_0 for which (6.6) is minimized and, hence, the optimal choice of γ_0.

$$\frac{\binom{n_0(1+\gamma_0)}{n_0}\left(\frac{|\Sigma_0|-1}{|\Sigma_0|}\right)^{n_0(1+\gamma_0)}\binom{n_1(1+\gamma_1)}{n_1}\left(\frac{|\Sigma_1|-1}{|\Sigma_1|}\right)^{n_1(1+\gamma_1)}}{\binom{n_0(1+\gamma_0)+1}{n_0}\left(\frac{|\Sigma_0|-1}{|\Sigma_0|}\right)^{n_0(1+\gamma_0)+1}\binom{n_1(1+\gamma_1)-1}{n_1}\left(\frac{|\Sigma_1|-1}{|\Sigma_1|}\right)^{n_1(1+\gamma_1)-1}}$$

$$
\begin{aligned}
&= \frac{\binom{n_0(1+\gamma_0)}{n_0}\binom{n_1(1+\gamma_1)}{n_1}\binom{|\Sigma_1|-1}{|\Sigma_1|}}{\binom{n_0(1+\gamma_0)+1}{n_0}\binom{|\Sigma_0|-1}{|\Sigma_0|}\binom{n_1(1+\gamma_1)-1}{n_1}} \\
&= \frac{\binom{n_0(1+\gamma_0)}{n_0}\binom{n_1(1+\gamma_1)}{n_1}}{\binom{n_0(1+\gamma_0)+1}{n_0}\binom{n_1(1+\gamma_1)-1}{n_1}} \times \frac{|\Sigma_0| \times (|\Sigma_1|-1)}{(|\Sigma_0|-1) \times |\Sigma_1|} \\
&= \frac{\frac{n_1(1+\gamma_1)}{n_1(1+\gamma_1)-n_1}}{\frac{n_0(1+\gamma_0)+1}{n_0(1+\gamma_0)+1-n_0}} \times \frac{|\Sigma_0| \times (|\Sigma_1|-1)}{(|\Sigma_0|-1) \times |\Sigma_1|} \\
&= \frac{\frac{n_1(1+\gamma_1)}{n_1\gamma_1}}{\frac{n_0(1+\gamma_0)+1}{n_0\gamma_0+1}} \times \frac{|\Sigma_0| \times (|\Sigma_1|-1)}{(|\Sigma_0|-1) \times |\Sigma_1|} \\
&= \frac{1+\frac{1}{\gamma_1}}{1+\frac{1}{\gamma_0+1/n_0}} \times \frac{1-\frac{1}{|\Sigma_1|}}{1-\frac{1}{|\Sigma_0|}}
\end{aligned}
$$

Therefore, the optimal choice of γ_0 would be one for which:

$$
\begin{aligned}
&\frac{1+\frac{1}{\gamma_1}}{1+\frac{1}{\gamma_0+1/n_0}} \times \frac{1-\frac{1}{|\Sigma_1|}}{1-\frac{1}{|\Sigma_0|}} = 1 \\
\Leftrightarrow \quad &\left(1+\frac{1}{\gamma_1}\right)\left(1-\frac{1}{|\Sigma_1|}\right) = \left(1+\frac{1}{\gamma_0+1/n_0}\right)\left(1-\frac{1}{|\Sigma_0|}\right)
\end{aligned}
$$

Given that families of codes with increasing block lengths n are considered, the term $\frac{1}{n_0} = \frac{1}{n \cdot \alpha(1-d/q)}$ vanishes. Thus, we are looking for a choice of γ_0 that satisfies

$$
\left(1+\frac{1}{\gamma_1}\right)\left(1-\frac{1}{|\Sigma_1|}\right) = \left(1+\frac{1}{\gamma_0}\right)\left(1-\frac{1}{|\Sigma_0|}\right).
$$

Putting this together with the equation $\alpha(1-\frac{d}{q})\gamma_0 + (1-\alpha)(1-\frac{d+1}{q})\gamma_1 = \gamma$ and solving the resulting system of equations analytically using computer software, the stated equation for the optimal choice of γ_0 is derived. □

6.3 Alphabet Size vs. the Gap from the Singleton Bound (Corollary 6.1.3)

Before providing the proof of Corollary 6.1.3, we start with an informal argument that shows the necessity of exponential dependence of alphabet size on ε^{-1}. Consider a communication of n symbols out of an alphabet of size q where adversary is allowed to delete $\delta < \frac{1}{2}$ fraction of symbols. Note that one can map symbols of the alphabet to binary strings of length $\log q$. Let adversary act as follows. He looks at the first $2n\delta$ symbols and among symbols whose binary representation start with zero and symbols whose binary representation start with one chooses the least frequent ones and removes them. This can be done by up to $n\delta$ symbol deletions and he can use the remainder of deletions arbitrarily. This

way, the receiver receives $n(1-\delta)$ symbols where only $q/2$ distinct symbols might appear in the first $n\delta$ ones. This means that receiver can essentially get $n(1-\delta)\log q - n\delta + 1$ bits of information which implies a rate upper bound of $1-\delta-\frac{1}{\log q}$. Therefore, to get a rate of $1-\delta-\varepsilon$, alphabet size has to satisfy the following.

$$\frac{1}{\log q} < \varepsilon \Rightarrow q > 2^{\varepsilon^{-1}}.$$

Proof of Corollary 6.1.3. According to Theorem 6.1.2, in order to obtain a family of codes of rate $1-\delta-\varepsilon$, the following condition should hold.

$$\begin{aligned}
1-\delta-\varepsilon &\leq q\delta'\left[(1-(d+1)/q)\left(1-\log_q\frac{1}{1-(d+1)/q}\right)\right] \\
&\quad +(1-q\delta')\left[(1-d/q)\left(1-\log_q\frac{1}{1-d/q}\right)\right] \\
\Rightarrow \varepsilon &\geq q\delta'\left[(1-(d+1)/q)\log_q\frac{1}{1-(d+1)/q}\right]+(1-q\delta')\left[(1-d/q)\log_q\frac{1}{1-d/q}\right] \\
\Rightarrow q &\geq 2^{\frac{1}{\varepsilon}\cdot q\delta'\left[(1-(d+1)/q)\log\frac{1}{1-(d+1)/q}\right]+(1-q\delta')\left[(1-d/q)\log\frac{1}{1-d/q}\right]} \geq 2^{\frac{f(\delta)}{\varepsilon}}
\end{aligned}$$

which finishes the proof. The only step that might need some clarification is the following inequality that bounds below the right hand term with some function that only depends on δ and is non-zero in $(0,1)$.

$$q\delta'\left[(1-(d+1)/q)\log\frac{1}{1-(d+1)/q}\right]+(1-q\delta')\left[(1-d/q)\log\frac{1}{1-d/q}\right] \geq f(\delta)$$

Note that the left hand side is the convex combination of the points that are obtained by evaluating function $g(x)=(1-x)\log\frac{1}{1-x}$ at multiples of $\frac{1}{q}$ (See Figure 6.6). We denote the left hand term by $g'(\delta,q)$.

We only need to find a function of δ that is non-zero in $(0,1)$ and is strictly smaller than such convex combinations for any q. One good candidate is the simple piece-wise linear function $f(\cdot)$ that consists of a segment from $(0,0)$ to $(0.5,0.2)$ and a segment from $(0.5,0.2)$ to $(1,0)$. Now, we show that for this choice of f, $g'(\delta,q)\geq f(\delta)$ for any q.

Note that $f(\delta)\leq g(\delta)$, therefore, for any $\frac{i}{q}\leq\delta\leq\frac{i+1}{q}$ that both $\frac{i}{q}$ and $\frac{i+1}{q}$ are smaller than $\frac{1}{2}$ or both are larger than $\frac{1}{2}$, $g'(\delta,q)\geq f(\delta)$. Further, for the case of $\frac{\lfloor q/2\rfloor}{q}\leq\delta\leq\frac{\lceil q/2\rceil}{q}$, we draw readers attention to the fact that constant 0.2 in the definition of $f(\cdot)$ has been chosen small enough so that $g'(1/2,3)\geq f(1/2)$ which implies that $g'(1/2,q)\geq f(1/2)$ for any q and consequently $g'(\delta,q)\geq f(\delta)$ for all $\frac{\lfloor q/2\rfloor}{q}\leq\delta\leq\frac{\lceil q/2\rceil}{q}$ (See Figure 6.6).

□

6.4 Analysis of Random Codes

6.4.1 Random Deletion Codes (Theorem 6.1.6)

Proof of Theorem 6.1.6. Let $\mathcal{C}$ be a random code that maps any $x\in[1..q]^{nR}$ to some member of $[1..q]^n$ denoted by $E_{\mathcal{C}}(x)$ that has been chosen uniformly at random. Note that

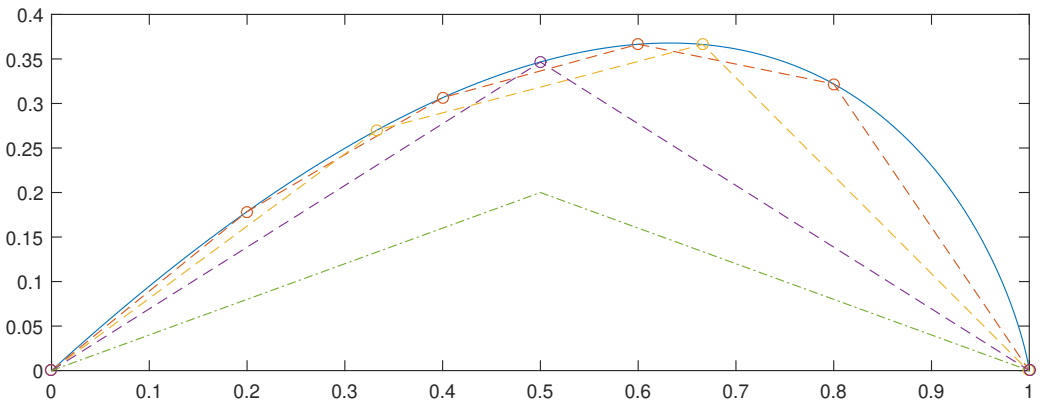


Figure 6.6: (I) Solid line: $g(\delta)$, (II) Dashed-dotted line: $f(\delta)$, (III) Dashed lines: $g'(\delta, q)$ for $q = 2, 3, 5$.

in a deletion channel, for any $y = E_{\mathcal{C}}(x)$ sent by Alice, the message $z \in [1..q]^{n(1-\delta)}$ Bob receives is necessarily a subsequence of y. The probability of a fixed $z \in [1..q]^{n(1-\delta)}$ being a subsequence of a random $y \in [1..q]^n$ is bounded above as follows.

$$\begin{aligned}
\Pr_y\{z \text{ is a substring of } y\} &= \sum_{1\le a_1<\cdots<a_{n(1-\delta)}\le n} q^{-n(1-\delta)}\left(1-\frac{1}{q}\right)^{a_{n(1-\delta)}-n(1-\delta)} && (6.10)\\
&= \sum_{l=n(1-\delta)}^{n}\binom{l}{n(1-\delta)}q^{-n(1-\delta)}\left(1-\frac{1}{q}\right)^{l-n(1-\delta)}\\
&\le n\delta\binom{n}{n(1-\delta)}q^{-n(1-\delta)}\left(1-\frac{1}{q}\right)^{n\delta} && (6.11)\\
&= n\delta 2^{nH(1-\delta)+o(n)}q^{-n}(q-1)^{n\delta}\\
&= q^{n\left((1-\delta)\log_q\frac{1}{1-\delta}+\delta\log_q\frac{1}{\delta}+\delta\log_q(q-1)-1+o(1)\right)} && (6.12)
\end{aligned}$$

Step (6.10) is obtained by conditioning the probability over the leftmost appearance of z in y as a subsequence. $a_1, \cdots, a_{n(1-\delta)}$ denote the positions in y where the first instance of z is occurred. Note that in such event, y_{a_i} has to be identical to z_i and symbol z_i cannot appear in $y[a_{i-1}+1, a_i-1]$. Hence, the probability of z appearing in $a_1, \cdots, a_{n(1-\delta)}$ as the leftmost occurrence is $\left(\frac{1}{q}\right)^{n(1-\delta)}\left(1-\frac{1}{q}\right)^{a_n-n(1-\delta)}$.

To verify (6.11), we show that the largest term of the summation in the previous step is attained at $l = n$ and bound the summation above by n times that term. To prove this, we simply show that the ratio of the consecutive terms of the summation is larger than

one.

$$\frac{\binom{l}{n(1-\delta)} q^{-n(1-\delta)} \left(1-\frac{1}{q}\right)^{l-n(1-\delta)}}{\binom{l-1}{n(1-\delta)} q^{-n(1-\delta)} \left(1-\frac{1}{q}\right)^{l-1-n(1-\delta)}} = \frac{l}{l-n(1-\delta)} \cdot \left(1-\frac{1}{q}\right) = \frac{1-\frac{1}{q}}{1-\frac{n(1-\delta)}{l}} \geq 1$$

where the last inequality follows from the fact that $\delta < 1 - \frac{1}{q} \Rightarrow \frac{1}{q} < 1 - \delta \Rightarrow \frac{1}{q} < \frac{n(1-\delta)}{l}$.

According to (6.12), for a random code $\mathcal{C}$ of rate R, the probability of $l+1$ codewords $E_{\mathcal{C}}(m_1), E_{\mathcal{C}}(m_2), \cdots, E_{\mathcal{C}}(m_{l+1})$ for $m_1, \cdots, m_{l+1} \in [1..q]^{nR}$ containing some fixed $z \in [1..q]^{n(1-\delta)}$ as a subsequence is bounded above as follows.

$$\Pr\{z \text{ is a subseq. of all } E_{\mathcal{C}}(m_1), \cdots\} \leq q^{n(l+1)\left((1-\delta)\log_q \frac{1}{1-\delta} + \delta \log_q \frac{1}{\delta} + \delta \log_q(q-1) - 1 + o(1)\right)}$$

Hence, by applying the union bound over all $z \in [1..q]^{n(1-\delta)}$, the probability of random code $\mathcal{C}$ not being l-list decodable can be bounded above as follows.

$$\begin{aligned} & \Pr_{\mathcal{C}}\left\{\exists z \in [1..q]^{n(1-\delta)}, m_1, \cdots, m_{l+1} \in [1..q]^{nR} \text{ s.t. } z \text{ is a subseq. of all } E_{\mathcal{C}}(m_1), \cdots\right\} \\ \leq \; & q^{n(1-\delta)} \left(q^{Rn}\right)^{l+1} q^{n(l+1)\left((1-\delta)\log_q \frac{1}{1-\delta} + \delta \log_q \frac{1}{\delta} + \delta \log_q(q-1) - 1 + o(1)\right)} \end{aligned}$$

As long as n's coefficient in the exponent of q is negative, this probability is less than one and drops exponentially to zero as n grows.

$$\begin{aligned} & n(1-\delta) + Rn(l+1) + n(l+1)\left((1-\delta)\log_q \frac{1}{1-\delta} + \delta \log_q \frac{1}{\delta} + \delta \log_q(q-1) - 1 + o(1)\right) < 0 \\ \Leftrightarrow \; & R < 1 - (1-\delta)\log_q \frac{1}{1-\delta} - \delta \log_q \frac{1}{\delta} - \delta \log_q(q-1) - \frac{1-\delta}{l+1} - o(1) \end{aligned} \tag{6.13}$$

Therefore, the random code $\mathcal{C}$ with any rate R that satisfies (6.13) is list-decodable with a list of size l with high probability.

We now proceed to prove the second side of Theorem 6.1.6. We will show that any family of random codes with rate $R > 1 - (1-\delta)\log_q \frac{1}{1-\delta} - \delta \log_q \frac{1}{\delta} - \delta \log_q(q-1)$ is not list decodable with high probability.

To see this, fix a received word $z \in [1..q]^{n(1-\delta)}$. Let X_i be the indicator random variable that indicates whether ith codeword contains z as a subsequence or not. Probability of $X_i = 1$ was calculated in (6.12) and gives that the expected number of codewords that contain z is

$$\mu = q^{nR} \cdot q^{n\left((1-\delta)\log_q \frac{1}{1-\delta} + \delta \log_q \frac{1}{\delta} + \delta \log_q(q-1) - 1 + o(1)\right)}.$$

Note that for $R > 1 - (1-\delta)\log_q \frac{1}{1-\delta} - \delta \log_q \frac{1}{\delta} - \delta \log_q(q-1)$ and large enough n, this number is exponentially large in terms of n. Further, as X_is are independent, Chernoff bound gives the following.

$$\Pr\{X > \mu/2\} \geq 1 - e^{-\mu/8}$$

Which implies that, with high probability, exponentially many codewords contain z and, therefore, the random code is not list-decodable. □

6.4.2 Random Insertion Codes (Theorem 6.1.7)

Proof of Theorem 6.1.7. We prove the claim by considering a random insertion code $\mathcal{C}$ that maps any $x \in [1..q]^{Rn}$ to some uniformly at random chosen member of $[1..q]^n$ denoted by $E_{\mathcal{C}}(x)$ and showing that it is possible to list-decode $\mathcal{C}$ with high probability.

Note that in an insertion channel, the original message sent by Alice is a substring of the message received on Bob's side. Therefore, a random insertion code $\mathcal{C}$ is l-list decodable if for any $z \in [1..q]^{(\gamma+1)n}$, there are at most l codewords of $\mathcal{C}$ that are subsequences of z. For some fixed $z \in [1..q]^{(\gamma+1)n}$, the probability of some uniformly at random chosen $y \in [1..q]^n$ being a substring of z can be bounded above as follows.

$$\begin{aligned}\Pr_y\{y \text{ is a subsequence of } z\} &\leq \binom{(\gamma+1)n}{n} q^{-n} \\ &= 2^{n(\gamma+1)H(\frac{1}{\gamma+1})+o(n)} q^{-n} \\ &= q^{n\left(\log_q(\gamma+1)+\gamma\log_q\frac{\gamma+1}{\gamma}-1+o(1)\right)}\end{aligned}$$

Therefore, for a random code $\mathcal{C}$ of rate R and any $m_1, \cdots, m_{l+1} \in [1..q]^{nR}$ and some fixed $z \in [1..q]^{n(\gamma+1)}$:

$$\Pr\{E_{\mathcal{C}}(m_1), \cdots, E_{\mathcal{C}}(m_{l+1}) \text{ are subsequences of } z\} \leq q^{n(l+1)\left(\log_q(\gamma+1)+\gamma\log_q\frac{\gamma+1}{\gamma}-1+o(1)\right)}$$

Hence, using the union bound over $z \in [1..q]^{n(\gamma+1)}$, for the random code $\mathcal{C}$:

$$\begin{aligned}&\Pr_{\mathcal{C}}\left\{\exists z \in [1..q]^{n(\gamma+1)}, m_1, \cdots, m_{l+1} \in q^{nR} \text{ s.t. } E_{\mathcal{C}}(m_1), \cdots, E_{\mathcal{C}}(m_{l+1}) \text{ are subseq. of } z\right\} \\ \leq\ & q^{n(\gamma+1)}\left(q^{Rn}\right)^{l+1} q^{n(l+1)\left(\log_q(\gamma+1)+\gamma\log_q\frac{\gamma+1}{\gamma}-1+o(1)\right)} \\ =\ & q^{n(\gamma+1)+Rn(l+1)+n(l+1)\left(\log_q(\gamma+1)+\gamma\log_q\frac{\gamma+1}{\gamma}-1+o(1)\right)} \qquad (6.14)\end{aligned}$$

As long as q's exponent in (6.14) is negative, this probability is less than one and drops exponentially to zero as n grows.

$$\begin{aligned}& n(\gamma+1)+Rn(l+1)+n(l+1)\left(\log_q(\gamma+1)+\gamma\log_q\frac{\gamma+1}{\gamma}-1+o(1)\right)<0 \\ \Leftrightarrow\ & R < 1-\log_q(\gamma+1)-\gamma\log_q\frac{\gamma+1}{\gamma}-\frac{\gamma+1}{l+1}+o(1) \qquad (6.15)\end{aligned}$$

Therefore, the family of random codes with any rate R that satisfies (6.15) is list-decodable with a list of size l with high probability. □

6.4.3 Random Insertion-Deletion Codes (Theorem 6.1.8)

Before Proceeding to the proof of Theorem 6.1.8, we provide a straightforward analysis of the size of the insertion-deletion ball.

Analysis of the Insertion-Deletion Ball

We start with a preliminary lemma.

Lemma 6.4.1 (From [Lev74])**.** *Let n, n_i, and q be positive integers and $S \in [q]^{n_i}$. Then, the number of strings that lie inside the insertion ball of radius n_i around S is*

$$|\mathcal{B}_i(S, n_i)| = \sum_{i=0}^{n_i} \binom{n+n_i}{i} (q-1)^i.$$

In the following, we give a simple bound on the size of the insertion-deletion ball.

Lemma 6.4.2. *Let $x \in [q]^n$, $\delta \in \left[0, 1 - \frac{1}{q}\right]$ and $\gamma \in [0, (q-1)(1-\delta)]$. The size of the insertion-deletion ball of insertion-radius γn and deletion-radius δn around x is no larger than*

$$\mathcal{B}(x, \gamma n, \delta n) \leq q^{n\left(H_q(\delta) + (1-\delta+\gamma)H_q\left(\frac{\gamma}{1-\delta+\gamma}\right) - \delta \log_q(q-1)\right) + o(n)}.$$

Proof.

$$\begin{aligned}
\mathcal{B}(x, \gamma n, \delta n) &= \sum_{x_0 \in \mathcal{B}_d(x,\delta)} |\mathcal{B}_i(x_0, \gamma)| \\
&\leq \binom{n}{\delta n} \sum_{i=0}^{\gamma n} \binom{n(1-\delta+\gamma)}{i} (q-1)^i && (6.16) \\
&\leq n\gamma \binom{n}{n\delta} \binom{n(1-\delta+\gamma)}{n\gamma} (q-1)^{\gamma n} && (6.17) \\
&= q^{n\left(H_q(\delta) - \delta\log_q(q-1) + (1-\delta+\gamma)H_q\left(\frac{\gamma}{1-\delta+\gamma}\right) - \gamma\log_q(q-1) + \gamma\log_q(q-1)\right) + o(n)} && (6.18) \\
&= q^{n\left(H_q(\delta) + (1-\delta+\gamma)H_q\left(\frac{\gamma}{1-\delta+\gamma}\right) - \delta\log_q(q-1)\right) + o(n)}
\end{aligned}$$

Note that (6.16) follows from Lemma 6.4.1 and (6.17) is true because the term in summation reaches its maximum when $i = n\gamma$. To see this, we test the ratio between the value of the term for two consecutive parameter values i and $i+1$:

$$\frac{\binom{n(1-\delta+\gamma)}{i+1}(q-1)^{i+1}}{\binom{n(1-\delta+\gamma)}{i}(q-1)^i} = \frac{n(1-\delta+\gamma)-i}{i+1}(q-1)$$

Note that $\frac{n(1-\delta+\gamma)-i}{i+1}(q-1) \geq 1 \Leftrightarrow \frac{n(1-\delta+\gamma)-i}{i+1} \geq \frac{1}{q-1} \Leftrightarrow iq + 1 \leq n(1-\delta+\gamma)(q-1)$. This holds for all $i \leq n\gamma$ because: $n\gamma q + 1 \leq n(1-\delta+\gamma)(q-1) \Leftrightarrow n\gamma < n(1-\delta)(q-1)$. Finally, (6.18) follows from the definition of the q-qry entropy function

$$H_q(x) = x\log_q(q-1) - x\log_q x - (1-x)\log_q(1-x)$$

and the equation $\binom{n}{np} = q^{n(H_q(p) - p\log_q(q-1)) + o(n)}$. □

Proof of Theorem 6.1.8

Take the random codeword $X \in [q]^n$ and some string $y \in [q]^{n(1-\delta+\gamma)}$ of length $n' = n(1-\delta+\gamma)$. Using Lemma 6.4.2, the probability of X being inside the insertion-deletion ball of deletion-radius $\delta' n' = \gamma n$ and insertion-radius $\gamma' n' = \delta n$ of y is

$$\begin{aligned}
\Pr\{X \in \mathcal{B}(y, \gamma' n', \delta' n')\} &\leq \frac{q^{n'\left(H_q(\delta')+(1-\delta'+\gamma')H_q\left(\frac{\gamma'}{1-\delta'+\gamma'}\right)-\delta'\log_q(q-1)\right)+o(n)}}{q^n} \\
&= \frac{q^{n(1-\delta+\gamma)\left(H_q\left(\frac{\gamma}{1-\delta+\gamma}\right)+\frac{1}{1-\delta+\gamma}H_q(\delta)-\frac{\gamma}{1-\delta+\gamma}\log_q(q-1)\right)+o(n)}}{q^n} \\
&= q^{n\left(H_q(\delta)+(1-\delta+\gamma)H_q\left(\frac{\gamma}{1-\delta+\gamma}\right)-\gamma\log_q(q-1)-1\right)+o(n)}
\end{aligned}$$

For the random code C with rate R to not be l-list decodable for some integer l, there has to exists some string $y \in [q]^{n(1-\delta+\gamma)}$ that can be obtained by $l+1$ codewords of C via δn deletions and γn insertions, i.e., codewords that lie in $\mathcal{B}(y, \delta n, \gamma n)$. Using the union bound over all $y \in [q]^{n(1-\delta+\gamma)}$, the probability of the existence of such y is at most.

$$\begin{aligned}
& q^{n(1-\delta+\gamma)}\left(q^{nR}\right)^{l+1}\left(q^{n\left(H_q(\delta)+(1-\delta+\gamma)H_q\left(\frac{\gamma}{1-\delta+\gamma}\right)-\gamma\log_q(q-1)-1\right)+o(n)}\right)^{l+1} \\
= \; & q^{n(l+1)\left(R+H_q(\delta)+(1-\delta+\gamma)H_q\left(\frac{\gamma}{1-\delta+\gamma}\right)-\gamma\log_q(q-1)-1+\frac{1-\delta+\gamma}{l+1}+o_l(1)\right)} \qquad (6.19)
\end{aligned}$$

Note that we used the trivial bound $\binom{q^{nR}}{l+1} \leq \left(q^{nR}\right)^{l+1}$ in the above calculation. Equation (6.19) implies that as long as

$$R < 1-(1-\delta+\gamma)H_q\left(\frac{\gamma}{1-\delta+\gamma}\right) - H_q(\delta) + \gamma\log_q(q-1),$$

for an appropriately large $l = O_{\gamma,\delta,q}(1)$, the exponent of (6.19) is negative and, therefore, the family of random codes is (γ, δ)-list-decodable with high probability. □

6.5 Rate Needed for Unique Decoding

For the sake of completeness, in this section we provide a formal argument for a claim made in Section 6.1.2. In the following claim, we assert that for every $\delta, \gamma > 0$ the rate of any insdel code that uniquely recovers from δ-fraction deletions and γ-fraction insertions is at most $1-(\gamma+\delta)$.

Claim 6.5.1. *If C is an insdel code of rate R that can recover from δ-fraction insertions and γ-fraction deletions with unique decoding, then $R \leq 1-(\delta+\gamma)$.*

Proof. Let $C \subseteq \Sigma^n$ be a code with q^k codewords, where $q = |\Sigma|$, for some integer $k > (1-(\delta+\gamma))n+1$. Consider the projection of codewords to the first $k-1$ coordinates. By the pigeonhole principle there must be two codewords $x, y \in C$ that agree on the first $k-1$ coordinates. Let $x = stu$ and $y = svw$ where s is of length $k-1$, t, v are of length γn,

and u, w are of length δn. Now consider the string stv: This can be obtained from either x or y by first deleting the last δn coordinates, and then inserting either t (for y) or v (for x). Thus no unique decoder can uniquely decode this code from δ fraction insertions and γ fraction deletions. □

The results of Chapter 3 in contrast show that given α and $\varepsilon > 0$ there is a single code C of rate $1 - \alpha - \varepsilon$ that can recover from δ fraction deletions and γ fraction insertions for any choice of δ, γ with $\gamma + \delta \leq \alpha$. Corollary 6.1.5 shows that any such result must have exponentially large alphabet size in ε though it does not rule out the possibility that there may exist specific choices of γ and δ for which smaller alphabets may suffice.

6.6 Missing Convexity Proof from Section 6.2.3

In this section, we show that the bivariate function

$$f(\gamma, \delta) = (1-\delta)\left[\left(1 + \frac{\gamma}{1-\delta}\right)\log_q \frac{q(1-\delta)}{\frac{\gamma}{1-\delta}+1} - \frac{\gamma}{1-\delta} \cdot \left(\log_q \frac{q(1-\delta)-1}{\frac{\gamma}{1-\delta}}\right)\right]$$

is convex. To prove the convexity, our general strategy is to show that the Hessian matrix of f is positive semi-definite. In order to do so, we take the following steps: We first characterize a domain D for $f(\gamma, \delta)$, over which we analyze the convexity. We then calculate the Hessian matrix of the function f, H_f. To show the positive semi-definiteness of H_f, we form its characteristic polynomial and then show that both of its solutions are real and non-negative – meaning that both eigenvalues of H_f are non-negative over the domain D. This would imply that H_f is positive semi-definite and, hence, f is convex over D.

Determining the domain D. Let us begin with describing the domain D. As stated in Section 6.1.1, we only consider the error rates that are within $\delta \in [0, 1 - 1/q]$ and $\gamma \in [0, q-1]$. Note that for any fixed value $\delta \in [0, 1-1/q)$, $f(\gamma = 0, \delta)$ is positive. We will show that as γ grows, the value of $f(\gamma, \delta)$ continuously drops until it reaches zero at $\gamma = (1-\delta)(q - q\delta - 1)$. This suggests that the domain D has to be defined as follows:

$$D = \left\{(\gamma, \delta) \mid 0 \leq \delta \leq 1 - \frac{1}{q}, 0 \leq \gamma \leq (1-\delta)(q - q\delta - 1)\right\}.$$

To show the claim above, we demonstrate two simple facts: (I) The partial derivative of f with respect to γ is negative within $0 \leq \gamma \leq (1-\delta)(q - q\delta - 1)$ and, (II) $f(\gamma, \delta) = 0$ for $\gamma = (1-\delta)(q - q\delta - 1)$.

To see claim (a), note that

$$\frac{\partial f}{\partial \gamma} = \log_q\left(\frac{(1-\delta)^2 q}{1-\delta+\gamma}\right) - \log_q\left(\frac{(1-\delta)(q-q\delta-1)}{\gamma}\right) = \log_q\left(\frac{q(1-\delta)\gamma}{(1-\delta+\gamma)(q-q\delta-1)}\right)$$

which is non-positive as long as

$$\begin{aligned}\frac{\partial f}{\partial \gamma} \leq 0 \quad &\Leftrightarrow \quad \frac{q(1-\delta)\gamma}{(1-\delta+\gamma)(q-q\delta-1)} < 1 \\ &\Leftrightarrow \quad q(1-\delta)\gamma \leq (1-\delta+\gamma)(q-q\delta-1) \\ &\Leftrightarrow \quad \gamma \leq (1-\delta)(q-q\delta-1)\end{aligned} \tag{6.20}$$

Note that (6.20) is valid since $1-\delta+\gamma \geq 0$ and $\delta \leq 1-\frac{1}{q} \Rightarrow q-q\delta-1 \geq 0$. One can also easily evaluate $f(\gamma,\delta)$ for $\gamma=(1-\delta)(q-q\delta-1)$ to confirm claim (b).

Hessian Matrix and Characteristic Polynomial. We now proceed to calculating the Hessian matrix of f and forming its characteristic polynomial.

$$\begin{aligned}H_f \quad &= \quad \begin{bmatrix} H_{1,1} & H_{1,2} \\ H_{2,1} & H_{2,2} \end{bmatrix} = \begin{bmatrix} \frac{\partial^2 f}{\partial \gamma^2} & \frac{\partial^2 f}{\partial \gamma \partial \delta} \\ \\ \frac{\partial^2 f}{\partial \delta \partial \gamma} & \frac{\partial^2 f}{\partial \delta^2} \end{bmatrix} \\ &= \quad \begin{bmatrix} \frac{1-\delta}{\gamma(1-\delta+\gamma)\log(q)} & \frac{\gamma+(1-\delta)^2 q}{(1-\delta)(1-\delta+\gamma)(q-q\delta-1)\log(q)} \\ \\ \frac{\gamma+(1-\delta)^2 q}{(1-\delta)(1-\delta+\gamma)(q-q\delta-1)\log(q)} & \frac{(1-\delta)^3 q^2(1-\delta+2\gamma)+(2(1-\delta)q-1)\left(\gamma^2-\gamma(1-\delta)-(1-\delta)^2\right)}{(1-\delta)^2(1-\delta+\gamma)(q-q\delta-1)^2\log(q)} \end{bmatrix}\end{aligned} \tag{6.21}$$

We prove semi-definiteness by deriving the characteristic polynomial of H_f. The eigenvalues of H_f are the roots of this polynomial.

$$\begin{aligned}\det(H_f-\lambda I)=0 \quad &\Leftrightarrow \quad \begin{vmatrix} H_{1,1}-\lambda & H_{1,2} \\ H_{2,1} & H_{2,2}-\lambda \end{vmatrix} = 0 \\ &\Leftrightarrow \quad (H_{1,1}-\lambda)(H_{2,2}-\lambda)-H_{1,2}H_{2,1}=0 \\ &\Leftrightarrow \quad \lambda^2-(H_{1,1}+H_{2,2})\lambda+(H_{1,1}H_{2,2}-H_{1,2}H_{2,1})=0\end{aligned} \tag{6.22}$$

To prove the semi-definiteness of H_f, we show that both of its eigenvalues are non-negative, or equivalently, the roots of the quadratic equation (6.22) are both non-negative. We remind the reader of the straightforward fact that in a quadratic equation of form $x^2-Sx+P=0$, S is the sum of the roots and P is their product. Therefore, to show that both roots are non-negative, we only need to show that S and P are both non-negative and that the roots are both real, i.e., $\Delta=S^2-4P \geq 0$.

1. $H_{1,1}+H_{2,2} \geq 0$
2. $H_{1,1}H_{2,2}-H_{1,2}H_{2,1} \geq 0$
3. $(H_{1,1}+H_{2,2})^2-4(H_{1,1}H_{2,2}-H_{1,2}H_{2,1}) \geq 0$

In the remainder of this section, we prove the three items listed above.

Proof of Item 1. Given (6.21), we have that

$$\begin{aligned} H_{1,1}+H_{2,2} &= \frac{1}{\log(q)}\cdot\left(\frac{1}{\gamma}+\frac{\gamma q^2}{(q-q\delta-1)^2}+\frac{\gamma(1-\gamma-\delta)}{(1-\delta)^2(1-\delta+\gamma)}\right) \\ &= \frac{1}{\log(q)}\cdot\left(\frac{1}{\gamma}+\frac{\gamma}{(1-1/q-\delta)^2}+\frac{\gamma}{(1-\delta)(1-\delta+\gamma)}-\frac{\gamma^2}{(1-\delta)^2(1-\delta+\gamma)}\right) \end{aligned} \tag{6.23}$$

Note that terms $\frac{1}{\gamma}$ and $\frac{\gamma}{(1-\delta)(1-\delta+\gamma)}$ are positive. Therefore, to prove that $H_{1,1}+H_{2,2}$ is non-negative, we show that

$$\frac{\gamma}{(1-1/q-\delta)^2}-\frac{\gamma^2}{(1-\delta)^2(1-\delta+\gamma)}\geq 0. \tag{6.24}$$

Note that

$$1-\delta-1/q<1-\delta \Rightarrow \frac{\gamma}{(1-1/q-\delta)^2}\geq\frac{\gamma}{(1-\delta)^2}. \tag{6.25}$$

Also, since $\delta\leq 1$, we have that $1-\delta+\gamma\geq\gamma\Rightarrow\frac{\gamma}{1-\delta+\gamma}\leq 1$. Thus, (6.25) holds even if one multiplies its right-hand side by $\frac{\gamma}{1-\delta+\gamma}$ which gives (6.24) and, thus, proves Item 1.

Proof of Item 2. Given (6.21), we have that

$$H_{1,1}H_{2,2}-H_{1,2}H_{2,1}=\frac{(1-\delta)^2(q-q\delta-1)^2-\gamma^2}{\gamma(1-\delta)^2(1-\delta+\gamma)(q-q\delta-1)^2\log^2(q)}.$$

Note that all terms in the denominator are positive. The numerator is positive as well since, as mentioned earlier, the domain D is defined only to include points (γ,δ) where $\gamma\leq(1-\delta)(q-q\delta-1)$.

Proof of Item 3. This claim can be simply shown to be true as follows:

$$(H_{1,1}+H_{2,2})^2-4(H_{1,1}H_{2,2}-H_{1,2}H_{2,1})=(H_{1,1}-H_{2,2})^2+4H_{1,2}H_{2,1}=(H_{1,1}-H_{2,2})^2+4H_{1,2}^2$$

The final term is trivially positive. Note that the last step follows from the fact that $H_{1,2}=H_{2,1}$. □

Chapter 7

Online Repositioning: Channel Simulations and Interactive Coding

In this chapter, we use an online repositioning algorithm for synchronization strings to present many new results related to reliable (interactive) communication over insertion-deletion channels.

We show how to hide the complications of synchronization errors in many applications by introducing very general *channel simulations* which efficiently transform an insertion-deletion channel into a regular symbol substitution channel with an error rate larger by a constant factor and a slightly smaller alphabet. Our channel simulations depend on the fact that, at the cost of increasing the error rate by a constant factor, synchronization strings can be decoded in a streaming manner that preserves linearity of time. Interestingly, we provide a lower bound showing that this constant factor cannot be improved to $1 + \varepsilon$, in contrast to what is achievable for error correcting codes. These channel simulations drastically and cleanly generalize the applicability of synchronization strings.

Using such channel simulations, we provide interactive coding schemes which simulate any interactive two-party protocol over an insertion-deletion channel. These results improve over the state-of-the-art interactive coding schemes of Braverman et al.[BGMO17] and Sherstov and Wu [SW19] which achieve a small constant rate and require exponential time computations with respect to computational and communication complexities. We provide the first computationally efficient interactive coding scheme for synchronization errors, the first coding scheme with a rate approaching one for small noise rates, and also the first coding scheme that works over arbitrarily small alphabet sizes. We also show tight connections between synchronization strings and edit-distance tree codes which allow us to transfer results from tree codes directly to edit-distance tree codes.

Finally, using our channel simulations, we provide an explicit binary insertion-deletion code achieving a rate of $1 - O(\sqrt{\delta \log(1/\delta)})$ that improves over the codes by Guruswami and Wang [GW17] in terms of the rate-distance trade-off.

7.1 Introduction

In this chapter, we show that one-way and interactive communication in the presence of synchronization errors can be reduced to the problem of communication in the presence of half-errors. We present a series of efficient channel simulations which allow two parties to communicate over a channel afflicted by synchronization errors as though they were communicating over a half-error channel with only a slightly larger error rate. This allows us to leverage existing coding schemes for robust communication over half-error channels in order to derive strong coding schemes resilient to synchronization errors.

One of the primary tools we use are *synchronization strings* which were introduced in Chapter 3 to design essentially optimal error correcting codes (ECCs) robust to synchronization errors. For every $\varepsilon > 0$, synchronization strings allow a sender to index a sequence of messages with an alphabet of size $\varepsilon^{-O(1)}$ in such a way that k synchronization errors are efficiently transformed into $(1+\varepsilon)k$ half-errors for the purpose of designing ECCs. Chapter 3 provide a black-box construction which transforms any ECC into an equally efficient ECC robust to synchronization errors. However, channel simulations and interactive coding in the presence of synchronization errors present a host of additional challenges that cannot be solved by the application of an ECC. Before we describe our results and techniques in detail, we begin with an overview of the well-known interactive communication model.

Interactive communication. Throughout this work, we study a scenario where there are two communicating parties, whom we call Alice and Bob. The two begin with some input symbols and wish to compute a function of their input by having a conversation. Their goal is to succeed with high probability while communicating as few symbols as possible. In particular, if their conversation would consist of n symbols in the noise-free setting, then they would like to converse for at most αn symbols, for some small α, when in the presence of noise. One might hope that Alice and Bob could correspond using error-correcting codes. However, this approach would lead to poor performance because if a party incorrectly decodes a single message, then the remaining communication is rendered useless. Therefore, only a very small amount of noise could be tolerated, namely less than the amount to corrupt a single message.

There are three major aspects of coding schemes for interactive communication that have been extensively studied in the literature. The first is the coding scheme's **maximum tolerable error-fraction** or, in other words, the largest fraction of errors for which the coding scheme can successfully simulate any given error-free protocol. Another important quality of coding schemes for interactive communication, as with one-way communication, is the **communication rate** or, equivalently, the amount of communication overhead in terms of the error fraction. Finally, the **efficiency** of interactive coding schemes have been of concern in the previous work.

Schulman initiated the study of error-resilient interactive communication, showing how to convert an arbitrary two-party interactive protocol to one that is robust to a $\delta = 1/240$ fraction of adversarial errors with a constant communication overhead [Sch92, Sch93]. Braverman and Rao increased the bound on the tolerable adversarial error rate to $\delta < 1/4$,

also with a constant communication overhead [BR14]. Brakerski et al. [BKN14] designed the first efficient coding scheme resilient to a constant fraction of adversarial errors with constant communication overhead. The above-mentioned schemes achieve a constant overhead no matter the level of noise. Kol and Raz were the first to study the trade-off between error fraction and communication rate [KR13]. Haeupler then provided a coding scheme with a communication rate of $1 - O(\sqrt{\delta \log \log(1/\delta)})$ over an adversarial channel [Hae14]. Further prior work has studied coding for interactive communication focusing on communication efficiency and noise resilience [GH17a, BE17, GH14] as well as computational efficiency [BTK12, BN13, BKN14, GMS11, GMS14, GH14]. Other works have studied variations of the interactive communication problem [GHS14, FGOS15, ERB16, AGS16, BNT+19].

The main challenge that *synchronization errors* pose is that they may cause the parties to become "out of sync." For example, suppose the adversary deletes a message from Alice and inserts a message back to her. Neither party will know that Bob is a message behind, and if this corruption remains undetected, the rest of the communication will be useless. In most state-of-the-art interactive coding schemes for symbol substitutions, the parties communicate normally for a fixed number of rounds and then send back and forth a series of checks to detect any symbol substitutions that may have occurred. One might hope that a synchronization error could be detected during these checks, but the parties may be out of sync while performing the checks, thus rendering them useless as well. Therefore, synchronization errors require us to develop new techniques.

Very little is known regarding coding for interactive communication in the presence of synchronization errors. A 2016 coding scheme by Braverman et al. [BGMO17], which can be seen as the equivalent of Schulman's seminal result, achieves a small constant communication rate while being robust against a $1/18 - \varepsilon$ fraction of errors. The coding scheme relies on edit-distance tree codes, which are a careful adaptation of Schulman's original tree codes [Sch93] for edit distance, so the decoding operations are not efficient and require exponential time computations. A recent work by Sherstov and Wu [SW19] closed the gap for maximum tolerable error fraction by introducing a coding scheme that is robust against $1/6 - \varepsilon$ fraction of errors which is the highest possible fraction of insertions and deletions under which any coding scheme for interactive communication can work. Both Braverman et al. [BGMO17] and Sherstov and Wu [SW19] schemes are of constant communication rate, over large enough constant alphabets, and inefficient. In this work we address the next natural questions which, as arose with ordinary interactive communication, are on finding interactive coding schemes that are computationally efficient or achieve super-constant communication efficiency.

7.1.1 Our results

We present very general channel simulations which allow two parties communicating over an insertion-deletion channel to follow any protocol designed for a regular symbol substitution channel. The fraction of errors on the simulated symbol substitution channel is only slightly larger than that on the insertion-deletion channel. Our channel simulations are made possible by synchronization strings. Crucially, at the cost of increasing the error

rate by a constant factor, synchronization strings can be decoded (i.e., repositioned) in a streaming manner which preserves the linearity of time. Note that the similar technique is used in Chapter 3 to transform synchronization errors into ordinary symbol substitutions as a stepping-stone to obtain insertion-deletion codes from ordinary error correcting codes in a black-box fashion. However, in the context of error correcting codes, there is no requirement for this transformation to happen in real time. In other words, in the study of insertion-deletion codes in Chapter 3, the entire message transmission is done and then the receiving party uses the entire message to transform the synchronization errors into symbol substitutions. In the channel simulation problem, this transformation is required to happen on the fly. Interestingly, we have found out that in the harder problem of channel simulation, the factor by which the number of synchronization errors increase by being transformed into substitution errors cannot be improved to $1 + o(1)$, in contrast to what is achievable for error correcting codes. This chapter exhibits the widespread applicability of synchronization strings and opens up several new use cases, such as coding for interactive communication over insertion-deletion channels. Namely, using synchronization strings, we provide techniques to obtain the following simulations of substitution channels over given insertion-deletion channels with binary and large constant alphabet sizes.

Theorem 7.1.1. *(Informal Statement of Theorems 7.3.3, 7.3.5, 7.3.11, and 7.3.13)*

(a) Suppose that n rounds of a one-way/interactive insertion-deletion channel over an alphabet Σ with a δ fraction of insertions and deletions are given. Using an ε-synchronization string over an alphabet Σ_{syn}, it is possible to simulate $n\,(1 - O_\varepsilon(\delta))$ rounds of a one-way/interactive substitution channel over Σ_{sim} with at most $O_\varepsilon\,(n\delta)$ symbols corrupted so long as $|\Sigma_{sim}| \times |\Sigma_{syn}| \leq |\Sigma|$.

(b) Suppose that n rounds of a binary one-way/interactive insertion-deletion channel with a δ fraction of insertions and deletions are given. It is possible to simulate $n(1 - \Theta(\sqrt{\delta \log(1/\delta)}))$ rounds of a binary one-way/interactive substitution channel with $\Theta(\sqrt{\delta \log(1/\delta)})$ fraction of substitution errors between two parties over the given channel.

Based on the channel simulations presented above, we present novel interactive coding schemes which simulate any interactive two-party protocol over an insertion-deletion channel.

We use our large alphabet interactive channel simulation along with constant-rate efficient coding scheme of Ghaffari and Haeupler [GH14] for interactive communication over symbol substitution channels to obtain a coding scheme for insertion-deletion channels that is efficient, has a constant communication rate, and tolerates up to $1/44 - \varepsilon$ fraction of errors. Note that despite the fact that this coding scheme fails to protect against the optimal $1/6 - \varepsilon$ fraction of synchronization errors as the recent work by Sherstov and Wu [SW19] does, it is an improvement over all previous work in terms of computational efficiency as it is the first efficient coding scheme for interactive communication over insertion-deletion channels.

Theorem 7.1.2. *For any constant $\varepsilon > 0$ and n-round alternating protocol Π, there is an efficient randomized coding scheme simulating Π in presence of $\delta = 1/44 - \varepsilon$ fraction of edit-corruptions with constant rate (i.e., in $O(n)$ rounds) and in $O(n^5)$ time that works with probability $1 - 2^{\Theta(n)}$. This scheme requires the alphabet size to be a large enough constant $\Omega_\varepsilon(1)$.*

Next, we use our small alphabet channel simulation and the substitution channel interactive coding scheme of Haeupler [Hae14] to introduce an interactive coding scheme for insertion-deletion channels. This scheme is not only computationally efficient, but also the first with super constant communication rate. In other words, this is the first coding scheme for interactive communication over insertion-deletion channels whose rate approaches one as the error fraction drops to zero. Our computationally efficient interactive coding scheme achieves a near-optimal communication rate of $1 - O(\sqrt{\delta \log(1/\delta)})$ and tolerates a δ fraction of errors. Besides computational efficiency and near-optimal communication rate, this coding scheme improves over all previous work in terms of alphabet size. As opposed to coding schemes provided by the previous work[BGMO17, SW19], our scheme does not require a large enough constant alphabet and works even for binary alphabets.

Theorem 7.1.3. *For sufficiently small δ, there is an efficient interactive coding scheme for fully adversarial binary insertion-deletion channels which is robust against δ fraction of edit-corruptions, achieves a communication rate of $1 - \Theta(\sqrt{\delta \log(1/\delta)})$, and works with probability $1 - 2^{-\Theta(n\delta)}$.*

We also utilize the channel simulations in one-way settings to provide efficient binary insertion-deletion codes correcting δ-fraction of synchronization errors–for δ smaller than some constant–with a rate of $1-\Theta(\sqrt{\delta \log(1/\delta)})$. This was an improvement in terms of rate-distance trade-off over the binary insertion-deletion codes by Guruswami and Wang [GW17] which were the state-of-the-art insertion-deletion code at the time of the publication of this result. The codes by Guruswami and Wang [GW17] achieve a rate of $1 - O(\sqrt{\delta} \log(1/\delta))$. Ever since, a couple of independent works by Haeupler [Hae19] and Cheng et al. [CJLW18] improved over our codes by introducing efficient binary codes with rate $1 - O(\delta \log^2 \frac{1}{\delta})$ via providing deterministic document exchange protocols.

Finally, we introduce a slightly improved definition of edit-distance tree codes, a generalization of Schulman's original tree codes defined by Braverman et al. [BGMO17]. We show that under our revised definition, edit-distance tree codes are closely related to synchronization strings. For example, edit-distance tree codes can be constructed by merging a regular tree code and a synchronization string. This transfers, for example, Braverman's sub-exponential time tree code construction [Bra12] and the candidate construction of Schulman [Sch03] from tree codes to edit-distance tree codes. Lastly, as a side note, we will show that with the improved definition, the coding scheme of Braverman et al. [BGMO17] can tolerate $1/10 - \varepsilon$ fraction of synchronization errors rather than $1/18 - \varepsilon$ fraction that the scheme based on their original definition did. This improved definition is independently observed by Sherstov and Wu [SW19].

7.1.2 The Organization of this Chapter

We start by reviewing basic definitions and concepts regarding interactive communication and synchronization strings in Section 7.2. Then we study channel simulations under various assumptions in Section 7.3. We use these channel simulations to obtain novel coding schemes for one-way and interactive communication in Sections 7.3 and 7.4. Finally, in Section 7.6, we discuss connections between synchronization strings, tree codes and edit-distance tree codes introduced by Braverman et al. [BGMO17].

7.2 Definitions and preliminaries

In this section, we define the channel models and communication settings considered in this work. We also review some important notations and lemmas related to synchronization strings from Chapter 3.

Error model and communication channels. In this chapter, we study two types of channels, *substitution channels* and *synchronization channels*. The former is one that suffers from symbol substitution and the latter from insertions and deletions.

In the one-way communication setting over an insertion-deletion channel, messages to the listening party may be inserted, and messages sent by the sending party may be deleted. The two-way channel requires a more careful setup. We emphasize that we cannot hope to protect against arbitrary insertions and deletions in the two-way setting because in the message-driven model, a single deletion could cause the protocol execution to "hang." Therefore, following the standard model of Braverman et al.'s work [BGMO17] that is employed in all other previous works on this problem [SW19], we restrict our attention to *edit corruptions*, which consist of a single deletion followed by a single insertion, which may be aimed at either party. Braverman et al. [BGMO17] provide a detailed discussion on their model and show that it is strong enough to generalize other natural models one might consider, including models that utilize clock time-outs to overcome the stalling issue.

In both the one- and two-way communication settings, we study *adversarial channels* with *error rate* δ. Our coding schemes are robust in both the *fully adversarial* and the *oblivious adversary* models. In the fully adversarial model, the adversary may decide at each round whether or not to interfere based on its state, its own randomness, and the symbols communicated by the parties so far. In the oblivious adversary model, the adversary must decide which rounds to corrupt in advance, and therefore independently of the communication history. A simple example of an oblivious adversary is the *random error channel*, where each round is corrupted with probability δ. In models we study, there is no pre-shared randomness between the parties.

Interactive and one-way communication protocols. In an *interactive protocol* Π over a channel with alphabet Σ, Alice and Bob begin with two inputs from Σ^* and then engage in n *rounds* of communication. In a single round, each party either listens for a message or sends a message, where this choice and the message, if one is generated, depends on the party's state, its input, and the history of the communication thus far. After the n

rounds, the parties produce an output. We study *alternating protocols*, where each party sends a message every other round and listens for a message every other round. In this message-driven paradigm, a party "sleeps" until a new message comes, at which point the party performs a computation and sends a message to the other party. Protocols in the interactive communication literature typically fall into two categories: *message-driven* and *clock-driven*. In the message-driven paradigm, a party "sleeps" until a new message comes, at which point the party performs a computation and sends a message to the other party. Meanwhile, in the clock-driven model, each party has a clock, and during a single tick, each party accepts a new message if there is one, performs a computation, and sends a message to the other party if he chooses to do so. In this work, we study the message-driven model, since in the clock-driven model, dealing with insertions and deletions is too easy. After all, an insertion would mean that one symbol is changed to another as in the case of the standard substitution model and a deletion would be detectable, as it would correspond to an erasure. In the presence of noise, we say that a protocol Π' *robustly simulates* a deterministic protocol Π over a channel C if given any inputs for Π, the parties can decode the transcript of the execution of Π on those inputs over a noise-free channel from the transcript of the execution of Π' over C.

Finally, we also study *one-way communication*, where one party sends all messages and the other party listens. Coding schemes in this setting are known as *error-correcting codes*.

String Notation. We recap the following definitions from Chapter 3 that were originally defined by [BGMO17]:

Definition 7.2.1 (String matching(Definition 3.2.1))**.** *Suppose that c and c' are two strings in Σ^*, and suppose that $*$ is a symbol not in Σ. Next, suppose that there exist two strings τ_1 and τ_2 in $(\Sigma \cup \{*\})^*$ such that $|\tau_1| = |\tau_2|$, $del(\tau_1) = c$, $del(\tau_2) = c'$, and $\tau_1[i] \approx \tau_2[i]$ for all $i \in \{1, \ldots, |\tau_1|\}$. Here, del is a function that deletes every $*$ in the input string and $a \approx b$ if $a = b$ or one of a or b is $*$. Then we say that $\tau = (\tau_1, \tau_2)$ is a* string matching *between c and c' (denoted $\tau : c \to c'$). We furthermore denote with $sc(\tau_i)$ the number of $*$'s in τ_i.*

Note that the *edit distance* between strings $c, c' \in \Sigma^*$ is exactly equal to $\min_{\tau: c \to c'} \{sc(\tau_1) + sc(\tau_2)\}$.

Definition 7.2.2 (Relative Suffix Pseudo-Distance or RSPD (Definition 3.5.20))**.** *Given any two strings $c, \tilde{c} \in \Sigma^*$, the* relative suffix pseudo-distance *between c and $\tilde{c}$ is*

$$\mathrm{RSPD}(c, \tilde{c}) = \min_{\tau: c \to \tilde{c}} \left\{ \max_{i=1}^{|\tau_1|} \left\{ \frac{sc(\tau_1[i, |\tau_1|]) + sc(\tau_2[i, |\tau_2|])}{|\tau_1| - i + 1 - sc(\tau_1[i, |\tau_1|])} \right\} \right\}$$

Indexing via Synchronization Strings and Intermediaries. We now review some important notions from previous chapters regarding synchronization strings long with defining the role of intermediaries that will be used later in our channel simulations. In short, synchronization strings allow communicating parties to protect against synchronization errors by indexing their messages. In this chapter, we think of this technique by introducing two intermediaries, C_A and C_B, that conduct the communication over the given insertion-deletion channel. C_A receives all symbols that Alice wishes to send to Bob, C_A sends

the symbols to C_B, and C_B communicates the symbols to Bob. C_A and C_B respectively handle the "indexing with synchronization strings" and "repositioning" procedures that is involved in keeping Alice and Bob in sync by helping C_B guess the actual index of symbols he receives. In this way, Alice and Bob communicate via C_A and C_B as though they were communicating over a half-error channel.

Let S be the synchronization string used by C_A to index the communication. Next, suppose that the adversary injects a total of $n\delta$ insertions and deletions, thus transforming the string S to the string S_τ. Here, $\tau = (\tau_1, \tau_2)$ is a string matching such that $del(\tau_1) = S$, $del(\tau_2) = S_\tau$, and for all $k \in [|\tau_1|] = [|\tau_2|]$,

$$(\tau_1[k], \tau_2[k]) = \begin{cases} (S[i], *) & \text{if } S[i] \text{ is deleted} \\ (S[i], S_\tau[j]) & \text{if } S_\tau[j] \text{ is successfully transmitted and sent as } S[i] \\ (*, S_\tau[j]) & \text{if } S_\tau[j] \text{ is inserted,} \end{cases}$$

where i is the index of $\tau_1[1, k]$ upon deleting the stars in $\tau_1[1, k]$, or in other words, $i = |del(\tau_1[1, k])|$ and similarly $j = |del(\tau_1[1, k])|$. We formally say that a symbol $S_\tau[j]$ is *successfully transmitted* if there exists a k such that $|del(\tau_2[1, k])| = j$ and $\tau_1[k] = \tau_2[k]$. It was not inserted or deleted by the adversary.

We remind the reader that C_A and C_B know the synchronization string S beforehand. The intermediary C_B will receive a set of transmitted indices $S_\tau[1], \ldots, S_\tau[n]$. Upon receipt of the jth transmitted index, for all $j \in [n]$, C_B "matches" $S_\tau[1, j]$ to a prefix $S[1, i]$ and therefore guesses the actual position of $S_\tau[j]$ as i. We call the algorithm that C_B runs to determine this matching an *(n, δ)-indexing algorithm.* The algorithm can also return a symbol $\top$ which represents an "I don't know" response. Formally, an (n, δ)-indexing solution was defined in Chapter 3 as follows.

Definition 7.2.3 ((n, δ)-indexing solution (Definition 3.2.2))**.** *The pair $(S, \mathcal{D}_S)$ consisting of a string $S \in \Sigma^n$ and an algorithm $\mathcal{D}_S$ is called an (n, δ)-indexing solution over alphabet Σ if for any set of $n\delta$ insertions and deletions corresponding to the string matching τ and altering the string S to a string S_τ, the algorithm $\mathcal{D}_S(S_\tau)$ outputs either $\top$ or an index between 1 and n for every symbol in S_τ.*

Recall that in Chapter 3, the symbol $S_\tau[j]$ is successfully transmitted if there exists an index k such that $|del(\tau_2[1, k])| = j$ and $\tau_1[k] = S[i]$ for some $i \in [n]$. It makes sense, then, to say that the algorithm correctly decodes $S_\tau[j]$ if it successfully recovers the index i. Indeed, we express this notion formally by saying that an (n, δ)-indexing solution $(S, \mathcal{D}_S)$ guesses the position of index j correctly under $\tau = (\tau_1, \tau_2)$ if $\mathcal{D}_S(S_\tau)$ outputs i and there exists a k such that $i = |del(\tau_1[1, k])|$, $j = |del(\tau_2[1, k])|$, $\tau_1[k] = S[i]$, and $\tau_2[k] = S_\tau[j]$. Notice that outputting $\top$ counts as an incorrect decoding. We now have the language to describe how well an (n, δ)-indexing solution performs. An algorithm has at most k misdecodings if for any τ corresponding to at most $n\delta$ insertions and deletions, there are at most k successfully transmitted and incorrectly repositioned symbols. An indexing solution is *streaming* if the decoded index for the ith element of the string S_τ only depends on $S_\tau[1, i]$.

We now recall one streaming indexing solution from Chapter 3 that is based on the following property of ε-synchronization strings.

Lemma 7.2.4 (Implied by Lemma 3.5.21)**.** *Let $S \in \Sigma^n$ be an ε-synchronization string and let $S_\tau[1, j]$ be a prefix of S_τ. Then there exists at most one index $i \in [n]$ such that the suffix distance between $S_\tau[1, j]$ and $S[1, i]$, denoted by* $\mathrm{RSPD}(S_\tau[1, j], S[1, i])$ *is at most $1 - \varepsilon$.*

This lemma suggests a simple (n, δ)-indexing solution given an input prefix $S_\tau[1, j]$: search over all prefixes of S for the one with the smallest RSPD from $S_\tau[1, j]$. We recall this algorithm outlined as Algorithm 4 from Chapter 3.

Theorem 7.2.5 (Theorems 3.5.23 and 3.5.24)**.** *Let $S \in \Sigma^n$ be an ε-synchronization string that is sent over an insertion-deletion channel with a δ fraction of insertions and deletions. There exists a streaming (n, δ)-indexing solution that returns a solution with $\frac{c_i}{1-\varepsilon} + \frac{c_d \varepsilon}{1-\varepsilon}$ misdecodings. The algorithm runs in time $O(n^5)$, spending $O(n^4)$ on each received symbol.*

7.3 Channel Simulations

In this section, we show how ε-synchronization strings can be used as a powerful tool to simulate substitution channels over insertion-deletion channels. In Section 7.5, we use these simulations to introduce coding schemes resilient to insertion-deletion errors.

We study the context where Alice and Bob communicate over an insertion-deletion channel, but via a blackbox channel simulation, they are able to run coding schemes that are designed for half-error channels. As we describe in Section 7.2, we discuss this simulation by introducing two intermediaries, C_A and C_B, that conduct the simulation by communicating over the given insertion-deletion channel.

7.3.1 One-way channel simulation over a large alphabet

Assume that Alice and Bob have access to n rounds of communication over a one-way insertion-deletion channel where the adversary is allowed to insert or delete up to $n\delta$ symbols. In this situation, we formally define a substitution channel simulation over the given insertion-deletion channel as follows:

Definition 7.3.1 (Substitution Channel Simulation)**.** *Let Alice and Bob have access to n rounds of communication over a one-way insertion-deletion channel with the alphabet Σ. The adversary may insert or delete up to $n\delta$ symbols. Intermediaries C_A and C_B* simulate *n' rounds of a substitution channel with alphabet Σ_{sim} over the given channel as follows. First, the adversary can insert a number of symbols into the insertion-deletion channel between C_A and C_B. Then for n' rounds $i = 1, \ldots, n'$, the following procedure repeats:*

1. *Alice gives $X_i \in \Sigma_{sim}$ to C_A.*

2. *Upon receiving X_i from Alice, C_A wakes up and sends a number of symbols (possibly zero) from the alphabet Σ to C_B through the given insertion-deletion channel. The adversary can delete any of these symbols or insert symbols* before, among, *or* after *them.*

3. *Upon receiving symbols from the channel, C_B wakes up and reveals a number of symbols (possibly zero) from the alphabet Σ_{sim} to Bob. We say all such symbols are* triggered *by X_i.*

Throughout this procedure, the adversary can insert or delete up to $n\delta$ symbols. However, C_B is required to reveal exactly n' symbols to Bob regardless of the adversary's actions. Let $\tilde{X}_1, \cdots, \tilde{X}_{n'} \in \Sigma_{sim}$ be the symbols revealed to Bob by C_B. This procedure successfully simulates n' rounds of a substitution channel with a δ' fraction of errors if for all but $n'\delta'$ elements i of the set $\{1, \ldots, n'\}$, the following conditions hold: 1) $\tilde{X}_i = X_i$; and 2) $\tilde{X}_i$ is triggered by X_i.

When $\tilde{X}_i = X_i$ and $\tilde{X}_i$ is triggered by X_i, we call $\tilde{X}_i$ an *uncorrupted symbol*. The second condition, that $\tilde{X}_i$ is triggered by X_i, is crucial to preserving linearity of time, which is the fundamental quality that distinguishes channel simulations from channel codings. It forces C_A to communicate each symbol to Alice as soon as it arrives. Studying channel simulations satisfying this condition is especially important in situations where Bob's messages depends on Alice's, and vice versa.

Conditions (1) and (2) also require that C_B conveys at most one uncorrupted symbol each time he wakes up. As the adversary may delete $n\delta$ symbols from the insertion-deletion channel, C_B will wake up at most $n(1-\delta)$ times. Therefore, we cannot hope for a substitution channel simulation where Bob receives more than $n(1-\delta)$ uncorrupted symbols. In the following theorem, we prove something slightly stronger: no deterministic one-way channel simulation can guarantee that Bob receives more than $n(1-4\delta/3)$ uncorrupted symbols and if the simulation is randomized, the expected number of uncorrupted transmitted symbols is at most $n(1-7\delta/6)$. This puts channel simulation in contrast to channel coding as one can recover $1-\delta-\varepsilon$ fraction of symbols there (as shown in Chapter 3).

Theorem 7.3.2. *Assume that n uses of a one-way insertion-deletion channel over an arbitrarily large alphabet Σ with a δ fraction of insertions and deletions are given. There is no deterministic simulation of a substitution channel over any alphabet Σ_{sim} where the simulated channel guarantees more than $n\,(1-4\delta/3)$ uncorrupted transmitted symbols. If the simulation is randomized, the expected number of uncorrupted transmitted symbols is at most $n(1-7\delta/6)$.*

Proof. Consider a simulation of n' rounds of a substitution channel on an insertion-deletion channel. Note that for any symbol that C_A receives, she will send some number of symbols to C_B. This number can be zero or non-zero and may also depend on the content of the symbol she receives. We start by proving the claim for deterministic simulations. Let $X_1, X_2, \cdots, X_{n'} \in \Sigma_{sim}$ and $X'_1, X'_2, \cdots, X'_{n'} \in \Sigma_{sim}$ be two possible sets of inputs that Alice may pass to C_A such that for any $1 \leq i \leq n'$, $X_i \neq X'_i$ and $Y_1, \cdots, Y_m \in \Sigma$ and let $Y'_1, \cdots, Y'_{m'} \in \Sigma$ be the symbols that C_A sends to C_B through the insertion-deletion channel as a result of receiving $\{X_i\}$ and $\{X'_i\}$ respectively.

Now, consider $Y_1, \cdots, Y_{n\delta}$. Let k be the number of C_A's input symbols which are required to trigger her to output $Y_1, \cdots, Y_{n\delta}$. We prove Theorem 7.3.2 in the following two cases:

1. $k \leq \frac{2n\delta}{3}$: In this case, $Y_1, \cdots, Y_{n\delta}$ will cause C_B to output at most $\frac{2n\delta}{3}$ uncorrupted symbols. If the adversary deletes $n\delta$ arbitrary elements among $Y_{n\delta+1}, \cdots, Y_m$, then C_B will receive $m - 2n\delta \leq n - 2n\delta$ symbols afterwards; Therefore, he cannot output more than $n - 2n\delta$ uncorrupted symbols as the result of receiving $Y_{n\delta+1}, \cdots, Y_m$. Hence, no simulation can guarantee $n(1-2\delta) + k < n\left(1 - \frac{4}{3}\delta\right)$ uncorrupted symbols or more.

2. $k > \frac{2n\delta}{3}$: Consider the following two scenarios:

   (a) Alice tries to convey $X_1, X_2, \cdots, X_{n'}$ to Bob using the simulation. The adversary deletes the first $n\delta$ symbols. Therefore, C_B receives $Y_{n\delta+1}, \cdots, Y_m$.

   (b) Alice tries to convey $X'_1, X'_2, \cdots, X'_{n'}$ to Bob using the simulation. The adversary inserts $Y_{n\delta+1}, \cdots, Y_{2n\delta}$ at the very beginning of the communication. Therefore, C_B receives $Y_{n\delta+1}, \cdots, Y_{2n\delta}, Y'_1, Y'_2, \cdots, Y'_{m'}$.

   Note that the first $n\delta$ symbols that C_B receives in these two scenarios are the same. Assume that C_B outputs k' symbols as the result of the first $n\delta$ symbols he receives. In the first scenario, the number of uncorrupted symbols C_B outputs as the result of his first $n\delta$ inputs is at most $\max\{0, k'-k\}$. Additionally, at most $m-2n\delta \leq n-2n\delta$ uncorrupted messages may be conveyed within the rest of the communication. In the second scenario, the number of uncorrupted communicated symbols is at most $n-k'$.

   Now, at least for one of these scenarios the number of guaranteed uncorrupted symbols in the simulation is

$$\begin{aligned} & \min\{n - 2n\delta + \max\{0, k' - k\}, n - k'\} \\ \leq\ & \max_{k'} \min\{n - 2n\delta + \max\{0, k' - k\}, n - k'\} \\ \leq\ & \max_{k'} \min\{n - 2n\delta + \max\{0, k' - 2n\delta/3\}, n - k'\} \\ =\ & n - 4n\delta/3 = n\left(1 - \frac{4}{3}\delta\right). \end{aligned}$$

This completes the proof for deterministic simulations. Now, we proceed to the case of randomized simulations.

Take an arbitrary input sequence $X_1, \cdots, X_{n'} \in \Sigma^{n'}$. Let K_X be the random variable that represents the number of C_A's input symbols which are required to trigger her to output her first $n\delta$ symbols to C_B. If $\Pr\{K_X \leq 2n\delta/3\} \geq \frac{1}{2}$, for any sequence $X_1, \cdots, \bar{X}_{n'}$ given to C_A by Alice, the adversary acts as follows. He lets the first $n\delta$ symbols sent by C_A pass through the insertion-deletion channel and then deletes the next $n\delta$ symbols that C_A sends to C_B. As in the deterministic case, if $K_X \leq 2n\delta/3$, the number of uncorrupted symbols conveyed to Bob cannot exceed $n(1 - 4\delta/3)$. Hence, the expected number of uncorrupted symbols in the simulation may be upper-bounded by:

$$\begin{aligned} \mathbb{E}[\text{Uncorrupted Symbols}] &\leq p \cdot n(1 - 4\delta/3) + (1-p) \cdot n(1-\delta) \\ &\leq n\left(1 - \frac{3+p}{3}\delta\right) \leq n\left(1 - \frac{3+1/2}{3}\delta\right) = n\left(1 - \frac{7}{6}\delta\right). \end{aligned}$$

Now, assume that $\Pr\{K_X \leq 2n\delta/3\} < \frac{1}{2}$. Take an arbitrary input $X'_1, X'_2, \cdots, X'_{n'} \in \Sigma_{sim}$ such that $X_i \neq X'_i$ for all $1 \leq i \leq n'$. Consider the following scenarios:

(a) Alice tries to convey $X_1, X_2, \cdots, X_{n'}$ to Bob using the simulation. The adversary removes the first $n\delta$ symbols sent by C_A. This means that C_B receives is $Y_{n\delta+1}, \cdots, Y_m$ where $Y_1, \cdots, Y_m$ is a realization of C_A's output distribution given $X_1, X_2, \cdots, X_{n'}$ as input.

(b) Alice tries to convey $X'_1, X'_2, \cdots, X'_{n'}$ to Bob using the simulation. The adversary mimics C_A and generates a sample of C_A's output distribution given $X_1, X_2, \cdots, X_{n'}$ as input. Let that sample be $Y_1, \cdots, Y_m$. The adversary inserts $Y_{n\delta+1}, \cdots, Y_{2n\delta}$ at the beginning and then lets the communication go on without errors.

Note that the distribution of the first $n\delta$ symbols that C_B receives, i.e., $Y_{n\delta+1}, \cdots, Y_{2n\delta}$, is the same in both scenarios. Let $K'_{X'}$ be the random variable that represents the number of symbols in C_B's output given that specific distribution over the symbols $Y_{n\delta+1}, \cdots, Y_{2n\delta}$. Now, according to the discussion we had for deterministic simulations, for the first scenario:

$$\begin{aligned}
\mathbb{E}[\text{Uncorrupted Symbols}] &\leq \mathbb{E}[n - 2n\delta + \max\{0, K'_{X'} - K_X\}] \\
&\leq n - 2n\delta + p \cdot \mathbb{E}[K'_{X'}] + (1-p)(\mathbb{E}[K'_{X'}] - 2n\delta/3) \\
&\leq n - 2n\delta + \mathbb{E}[K'_{X'}] - 2(1-p)n\delta/3
\end{aligned}$$

and for the second one:

$$\mathbb{E}[\text{Uncorrupted Symbols}] \leq \mathbb{E}[n - K'_{X'}] \leq n - \mathbb{E}[K'_{X'}].$$

Therefore, in one of the above-mentioned scenarios

$$\begin{aligned}
\mathbb{E}[\text{Uncorrupted Symbols}] &\leq \min\{n - 2n\delta + \mathbb{E}[K'_{X'}] - 2(1-p)n\delta/3, n - \mathbb{E}[K'_{X'}]\} \\
&\leq \max_{\gamma} \min\{n - 2n\delta + \gamma - 2(1-p)n\delta/3, n - \gamma\} \\
&= n\left(1 - \frac{4-p}{3}\delta\right) \\
&< n\left(1 - \frac{4-1/2}{3}\delta\right) \\
&= n\left(1 - \frac{7}{6}\delta\right).
\end{aligned}$$

Therefore, for any randomized simulation, there exists an input and a strategy for the adversary where

$$\mathbb{E}[\text{Uncorrupted Symbols}] \leq n\left(1 - \frac{7}{6}\delta\right).$$

□

We now provide a channel simulation using ε-synchronization strings. Every time C_A receives a symbol from Alice (from an alphabet Σ_{sim}), C_A appends a new symbol from a predetermined ε-synchronization string over an alphabet Σ_{syn} to Alice's symbol and sends it as one message through the channel. On the other side of channel, suppose that C_B has already revealed some number of symbols to Bob. Let $\mathtt{I}_B$ be the index of the next symbol C_B expects to receive. In other words, suppose that C_B has already revealed $\mathtt{I}_B - 1$ symbols to Bob. Upon receiving a new symbol from C_A, C_B uses the part of the message coming from the synchronization string to guess the index of the message Alice sent. We will refer to this decoded index as $\tilde{\mathtt{I}}_A$ and its actual index as $\mathtt{I}_A$. If $\widetilde{\mathtt{I}}_A < \mathtt{I}_B$, then C_B reveals nothing to Bob and ignores the message he just received. Meanwhile, if $\widetilde{\mathtt{I}}_A = \mathtt{I}_B$, then C_B reveals Alice's message to Bob. Finally, if $\widetilde{\mathtt{I}}_A > \mathtt{I}_B$, then C_B sends a dummy symbol to Bob and then sends Alice's message.

Given that the adversary can insert or delete up to $n\delta$ symbols, if C_A sends n symbols, then C_B may receive between $n - n\delta$ and $n + n\delta$ symbols. We do not assume the parties have access to a clock, so we must prevent C_B from stalling after C_A has sent all n messages. Therefore, C_B only listens to the first $n(1-\delta)$ symbols it receives.

The protocols of C_A and C_B are more formally described in Algorithm 6 . Theorem 7.3.3 details the simulation guarantees.

Algorithm 6 Simulation of a one-way constant alphabet channel

Initialize parameters: $S \leftarrow$ ε-synchronization string of length n

if C_A **then**
    Reset Status: $\mathtt{i} \leftarrow 0$
    for n iterations **do**
        Get m from Alice, send $(m, S[\mathtt{i}])$ to C_B, and increment $\mathtt{i}$ by 1.

if C_B **then**
    Reset Status: $\mathtt{I}_B \leftarrow 0$
    for $n(1-\delta)$ iterations **do**
        Receive $(\tilde{m}, \tilde{s})$ sent by C_A and set $\tilde{\mathtt{I}}_A \leftarrow$ Synchronization string decode$(\tilde{s}, S)$
        if $\tilde{\mathtt{I}}_A = \mathtt{I}_B$ **then**
            Send $\tilde{m}$ to Bob and increment $\mathtt{I}_B$.
        if $\tilde{\mathtt{I}}_A < \mathtt{I}_B$ **then**
            Continue
        if $\tilde{\mathtt{I}}_A > \mathtt{I}_B$ **then**
            Send a dummy symbol and then $\tilde{m}$ to Bob, then increment $\mathtt{I}_B$ by 2.

Theorem 7.3.3. *Assume that n uses of a one-way insertion-deletion channel over an alphabet Σ with a δ fraction of insertions and deletions are given. Using an ε-synchronization string over an alphabet Σ_{syn}, it is possible to simulate $n(1-\delta)$ rounds of a one-way substitution channel over Σ_{sim} with at most $2n\delta(2 + (1-\varepsilon)^{-1})$ symbols substituted so long as $|\Sigma_{sim}| \times |\Sigma_{syn}| \leq |\Sigma|$ and $\delta < 1/7$.*

Proof. Let Alice and Bob use n rounds of an insertion-deletion channel over alphabet Σ as sender and receiver respectively. We describe the simulation as being coordinated by two intermediaries C_A and C_B, who act according to Algorithm 6.

In order to find a lower-bound on the number of rounds of the simulated communication that remain uncorrupted, we upper-bound the number of rounds that can be corrupted. To this end, let the adversary insert k_i symbols and delete k_d symbols from the communication. Clearly, the k_d deleted symbols do not pass across the channel. Also, each of the k_i inserted symbols may cause C_B to change I_B. We call these two cases *error-bad* incidents. Further, $n\delta + k_i - k_d$ symbols at the end of the communication are not conveyed to Bob as we truncate the communication at $n(1-\delta)$. Moreover, according to Theorem 7.2.5, $\frac{k_i}{1-\varepsilon} + \frac{k_d\varepsilon}{1-\varepsilon}$ successfully transmitted symbols may be misdecoded upon their arrival. We call such incidents *decoding-bad* incidents. Finally, we need to count the number of successfully transmitted symbols whose indexes are decoded correctly ($\tilde{I}_A = I_A$) but do not get conveyed to Bob because $\tilde{I}_A \notin \{I_B, I_B + 1\}$, which we call *zero-bad* incidents. Zero-bad incidents happen only if $|I_A - I_B| \neq 0$. To count the number of zero-bad incidents, we have to analyze how I_A and I_B change in any of the following cases:

Cases when $I_A > I_B$	I_A	I_B	$I_A - I_B$
Deletion by the Adversary	+1	0	+1
Insertion by the Adversary	0	0,+1,+2	0, -1, -2
Correctly Transmitted but Misdecoded	+1	0,+1,+2	-1, 0, +1
Correctly Transmitted and Decoded	+1	+2	-1
Cases when $I_A < I_B$	I_A	I_B	$I_B - I_A$
Deletion by the Adversary	+1	0	-1
Insertion by the Adversary	0	0,+1,+2	0, +1, +2
Correctly Transmitted but Misdecoded	+1	0,+1,+2	-1, 0, +1
Correctly Transmitted and Decoded	+1	0	-1
Cases when $I_A = I_B$	I_A	I_B	$I_B - I_A$
Deletion by the Adversary	+1	0	-1
Insertion by the Adversary	0	0,+1,+2	0, +1, +2
Correctly Transmitted but Misdecoded	+1	0,+1,+2	-1, 0, +1
Correctly Transmitted and Decoded	+1	+1	0

Table 7.1: How I_A and I_B change in different scenarios.

Note that any insertion may increase $|I_A - I_B|$ by up to 2 units and any misdecoding or deletion may increase $|I_A - I_B|$ by up to 1 unit. Therefore, $|I_A - I_B|$ may be increased $2k_i + k_d + \frac{k_i}{1-\varepsilon} + \frac{k_d\varepsilon}{1-\varepsilon}$ throughout the algorithm. However, as any successfully transmitted and correctly decoded symbol decreases this variable by at least one, there are at most $2k_i + k_d + \frac{k_i}{1-\varepsilon} + \frac{k_d\varepsilon}{1-\varepsilon}$ zero-bad incidents, i.e. successfully transmitted and correctly decoded symbols that are not conveyed successfully in the simulated substitution channel.

Hence, the following is an upper-bound on the number of symbols that may not remain

uncorrupted in the simulated channel:

$$\begin{aligned} & \#(\text{error-bad}) + \#(\text{decoding-bad}) + \#(\text{zero-bad}) + \#(\text{truncated symbols}) \\ \leq\ & k_d + \left[\frac{k_i}{1-\varepsilon} + \frac{k_d\varepsilon}{1-\varepsilon}\right] + \left[2k_i + k_d + \frac{k_i}{1-\varepsilon} + \frac{k_d\varepsilon}{1-\varepsilon}\right] + [n\delta + k_i - k_d] \\ =\ & n\delta + k_d\left(1 + \frac{2\varepsilon}{1-\varepsilon}\right) + k_i\left(3 + \frac{2}{1-\varepsilon}\right) \leq n\delta\left(4 + \frac{2}{1-\varepsilon}\right). \end{aligned}$$

As error fraction shall not exceed one, the largest δ for which this simulation works is as follows.

$$\left.\frac{n\delta\left(4+\frac{2}{1-\varepsilon}\right)}{n(1-\delta)}\right|_{\varepsilon=0} = \frac{6\delta}{1-\delta} < 1 \Leftrightarrow \delta < \frac{1}{7}$$

□

7.3.2 Interactive channel simulation over a large alphabet

We now turn to channel simulations for interactive channels. As in Section 7.3.1, we formally define a substitution interactive channel simulation over a given insertion-deletion interactive channel. We then use synchronization strings to present one such simulation.

Definition 7.3.4 (Substitution Interactive Channel Simulation)**.** *Let Alice and Bob have access to n rounds of communication over an interactive insertion-deletion channel with alphabet Σ. The adversary may insert or delete up to $n\delta$ symbols. The simulation of an interactive substitution channel is performed by a pair of intermediaries C_A and C_B where Alice communicates with C_A, C_A interacts over the given insertion-deletion channel with C_B, and C_B communicates with Bob. More precisely, C_A and C_B* simulate *n' rounds of a substitution interactive channel with alphabet Σ_{sim} over the given channel as follows. The communication starts when Alice gives a symbol from Σ_{sim} to C_A. Then Alice, Bob, C_A, and C_B continue the communication as follows:*

1. *Whenever C_A receives a symbol from Alice or C_B, he either reveals a symbol from Σ_{sim} to Alice or sends a symbol from Σ through the insertion-deletion channel to C_B.*
2. *Whenever C_B receives a symbol from Bob or C_A, he either reveals a symbol from Σ_{sim} to Bob or send a symbols from Σ through the insertion-deletion channel to C_A.*
3. *Whenever C_B reveals a symbol to Bob, Bob responds with a new symbol from Σ_{sim}.*
4. *Whenever C_A reveals a symbol to Alice, Alice responds with a symbol in Σ_{sim} except for the $\frac{n'}{2}$th time.*

Throughout this procedure, the adversary can inject up to $n\delta$ edit corruptions. However, regardless of the adversary's actions, C_A and C_B have to reveal exactly $n'/2$ symbols to Alice and Bob respectively.

Let $X_1, \ldots, X_{n'}$ be the symbols Alice gives to C_A and $\tilde{X}_1, \ldots, \tilde{X}_{n'} \in \Sigma_{sim}$ be the symbols C_B reveals to Bob. Similarly, Let $Y_1, \ldots, Y_{n'}$ be the symbols Bob gives to C_B and

Algorithm 7 Simulation of a substitution channel using an insertion-deletion channel with a large alphabet: C_A's procedure

$\Pi \leftarrow n$-round interactive coding scheme over a substitution channel to be simulated
Initialize parameters: $S \leftarrow \varepsilon$-synchronization string of length $n/2$
$\mathtt{I}_A \leftarrow 0$
for $n/2 - n\delta\left(1 + \frac{1}{1-\varepsilon}\right)$ iterations **do**
    Get m from Alice, send $(m, S[\mathtt{I}_A])$ to C_B, and increment $\mathtt{I}_A$ by 1.
    Get $(\tilde{m}, \tilde{S})$ from C_B and send $\tilde{m}$ to Alice.
Commit.
for $n\delta\left(1 + \frac{1}{1-\varepsilon}\right)$ iterations **do**
    Send $(0, 0)$ to C_B, and increment $\mathtt{I}_A$ by 1.
    Get $(\tilde{m}, \tilde{S})$ from C_B.

$\tilde{Y}_1, \ldots, \tilde{Y}_{n'} \in \Sigma_{sim}$ be the symbols C_A reveals to Alice. We call each pair of tuples $(X_i, \tilde{X}_i)$ and $(Y_i, \tilde{Y}_i)$ a round *of the simulated communication. We call a round* corrupted *if its elements are not equal. This procedure successfully simulates n' rounds of a substitution interactive channel with a δ' fraction of errors if for all but $n'\delta'$ of the rounds are corrupted.*

Theorem 7.3.5. *Assume that n uses of an interactive insertion-deletion channel over an alphabet Σ with a δ fraction of insertions and deletions are given. Using an ε-synchronization string over an alphabet Σ_{syn}, it is possible to simulate $n - 2n\delta(1+(1-\varepsilon)^{-1})$ uses of an interactive substitution channel over Σ_{sim} with at most a $\frac{2\delta(5-3\varepsilon)}{1-\varepsilon+2\varepsilon\delta-4\delta}$ fraction of symbols corrupted so long as $|\Sigma_{sim}| \times |\Sigma_{syn}| \leq |\Sigma|$ and $\delta < 1/14$.*

Proof. Suppose that Alice and Bob want to communicate a total of $n - 2n\delta\left(1 + \frac{1}{1-\varepsilon}\right)$ symbols over the simulated substitution channel, and that this channel is simulated by the intermediaries C_A and C_B, who are communicating via a total of n uses of an insertion-deletion channel. We will show later on that both parties have the chance to commit before the other has sent $n/2$ messages over the insertion-deletion channel. We say an intermediary *commits* when it finishes simulating the channel for its corresponding party, i.e., when it sends the last simulated symbol out. Intermediaries may commit and yet carry on exchanging symbols over the channel so that the other intermediary finishes its simulation as well. An intermediary may stall by waiting for receiving symbols from the channel but the nature of simulation necessitates the intermediaries not to stall before they commit.

To analyze this simulation, we categorize the bad events that could occur as follows. We say that C_A takes a *step* when it sends a message, receives a message, and completes its required communication with Alice. We say that C_A's step is *good* if the message C_A receives is an uncorrupted response to its previous outgoing message and C_B correctly decodes the index that C_A sends. Figure 7.1 illustrates a sequence of steps where only the first is good.

If C_A has a good step when $\mathtt{I}_A = \mathtt{I}_B$ and neither party has committed, then Alice and Bob are guaranteed to have an error-free round of communication. We lower bound the

Algorithm 8 Simulation of a substitution channel using an insertion-deletion channel with a large alphabet: C_B's procedure

$\Pi \leftarrow n$-round interactive coding scheme over a substitution channel to be simulated
Initialize parameters: $S \leftarrow \varepsilon$-synchronization string of length $n/2$
$\mathtt{I}_B \leftarrow 0$
for $n/2$ iterations **do**
    Receive $(\tilde{m}, \tilde{s})$ from C_A.
    $\tilde{\mathtt{I}}_A \leftarrow$ Synchronization string decode$(\tilde{s}, S)$.
    if Committed **then**
        Send a dummy message to C_A.
    else if $\tilde{\mathtt{I}}_A = \mathtt{I}_B$ **then**
        Send $\tilde{m}$ to Bob and increment $\mathtt{I}_B$ by 1.
        Receive m from Bob and send $(m, 0)$ to C_A.
    else if $\tilde{\mathtt{I}}_A < \mathtt{I}_B$ **then**
        Send a dummy message to C_A.
    else if $\tilde{\mathtt{I}}_A > \mathtt{I}_B$ **then**
        Send a dummy message to Bob.
        Send $\tilde{m}$ to Bob and increment $\mathtt{I}_B$ by 2.
        Receive m from Bob and send $(m, 0)$ to C_A.
    If $\mathtt{I}_B = n/2 - n\delta\left(1 + \frac{1}{1-\varepsilon}\right)$, commit.

total number of Alice and Bob's error-free rounds of communication by lower bounding the number of good steps that C_A takes when $\mathtt{I}_A = \mathtt{I}_B$. The total number of good steps C_A takes is $S = n/2 - n\delta\left(1 + \frac{1}{1-\varepsilon}\right) - d$, where d is the number of rounds there are before C_A commits but after C_B commits, if C_B commits first. If a step is not good, then we say it is *bad*. Specifically, we say that it is *commit-bad* if C_B commits before C_A. We say that it is *decoding-bad* if neither party has committed, C_B receives an uncorrupted message from C_A, but C_B does not properly decode the synchronization string index. Otherwise, we say that a bad step is *error-bad*. Since every good step corresponds to a message sent by both Alice and Bob, we may lower bound the number of error-free messages sent over the substitution channel by

$$2\Bigg(\frac{n}{2} - n\delta\left(1 + \frac{1}{1-\varepsilon}\right) - d - \#(\text{error-bad steps}) - \#(\text{decoding-bad steps}) \\ -\#(\text{good steps when } \mathtt{I}_A \neq \mathtt{I}_B)\Bigg).$$

In order to lower bound this quantity, we now upper bound d, the number of error-bad and decoding-bad steps, and the number of good steps when $\mathtt{I}_A \neq \mathtt{I}_B$.

We claim that there are at most $n\delta$ error-bad steps since a single adversarial injection could cause C_A's current step to be bad, but that error will not cause the next step to be bad. Next, we appeal to Theorem 7.2.5 to bound the number of decoding-bad steps. The cited theorem guarantees that if an ε-synchronization string of length m is sent over

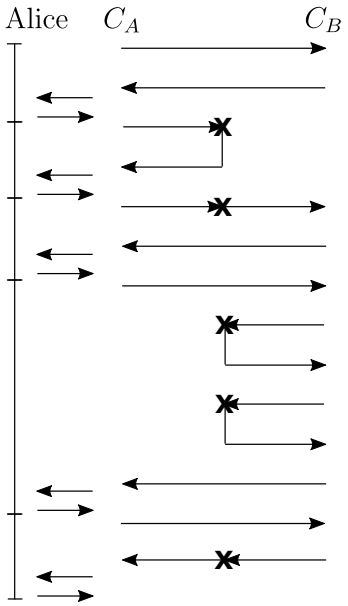


Figure 7.1: Illustration of five steps where only the first is good.

an insertion-deletion channel with a δ' fraction of errors, then the receiver will decode the index of the received symbol correctly for all but $\frac{\delta' m}{1-\varepsilon}$ symbols. In this scenario, $m = n/2$ and $\delta' = n\delta/(n/2) = 2\delta$ since an error over the insertion-deletion channel can cause at most one error in the synchronization string transmission. Therefore, there are at most $n\delta/(1-\varepsilon)$ decoding-bad steps.

Now, a single error, be it an adversarial injection or synchronization strings improper decoding, may cause $|\mathtt{I}_B - \mathtt{I}_A|$ to increase by at most two since a single error may cause C_B to simulate Step 16 of Algorithm 8 and increment $\mathtt{I}_B$ by two, while $\mathtt{I}_A$ does not change. Further, if an error does not occur but C_B incorrectly decodes the synchronization symbol C_A sends, then $|\mathtt{I}_B - \mathtt{I}_A|$ may increase by at most one, since C_B might increase $\mathtt{I}_B$ by two or zero while C_A increases $\mathtt{I}_A$ by one. Meanwhile, if $|\mathtt{I}_A - \mathtt{I}_B| \geq 1$ and C_A takes a good step, then this difference will decrease by at least 1. In total, over the course of the computation, since there are at most $n\delta$ adversarial injections and $\frac{n\delta}{1-\varepsilon}$ synchronization string improper decodings, we have that $|\mathtt{I}_B - \mathtt{I}_A|$ increases by at most $2n\delta + \frac{n\delta}{1-\varepsilon}$, which means that C_A will take at most $2n\delta + \frac{n\delta}{1-\varepsilon}$ good steps where $\mathtt{I}_B \neq \mathtt{I}_A$. We may bound this number of good steps a bit tighter as follows. Let $\mathtt{I}_A^*$ and $\mathtt{I}_B^*$ be the values of $\mathtt{I}_A$ and $\mathtt{I}_B$ when the first of C_A or C_B commits. If $|\mathtt{I}_A^* - \mathtt{I}_B^*| > 0$, then each party only had at most $2n\delta + \frac{n\delta}{1-\varepsilon} - |\mathtt{I}_A^* - \mathtt{I}_B^*|$ good steps where $\mathtt{I}_B \neq \mathtt{I}_A$.

Finally, we must upper bound the number d of rounds there are before C_A commits but after C_B commits if C_B commits first. Assuming C_B commits first, it must be that $\mathtt{I}_B^* = \frac{n}{2} - n\delta\left(1 + \frac{1}{1-\varepsilon}\right)$. Therefore, the number of steps before C_A commits is

$$d = \frac{n}{2} - n\delta\left(1 + \frac{1}{1-\varepsilon}\right) - \mathtt{I}_A^* = |\mathtt{I}_A^* - \mathtt{I}_B^*|\,.$$

We have shown that the number of error-free messages sent over the substitution channel

is at least

$$\begin{aligned}
&2\Bigg(\frac{n}{2} - n\delta\left(1+\frac{1}{1-\varepsilon}\right) - d - \#(\text{error-bad steps}) - \#(\text{decoding-bad steps}) \\
&-\#(\text{good steps when } \mathtt{I}_A \neq \mathtt{I}_B)\Bigg) \\
&\leq\ 2\left(\frac{n}{2} - n\delta\left(1+\frac{1}{1-\varepsilon}\right) - |\mathtt{I}_B^* - \mathtt{I}_A^*| - n\delta - \frac{n\delta}{1-\varepsilon} - \left(2n\delta + \frac{n\delta}{1-\varepsilon} - |\mathtt{I}_A^* - \mathtt{I}_B^*|\right)\right) \\
&=\ n - 8n\delta - \frac{6n\delta}{1-\varepsilon}.
\end{aligned}$$

Since Alice and Bob send a total of $n - 2n\delta\left(1+\frac{1}{1-\varepsilon}\right)$ messages over the simulated channel, the error rate δ_s over the simulated channel is at most $1 - \frac{n-8n\delta-\frac{6n\delta}{1-\varepsilon}}{n-2n\delta\left(1+\frac{1}{1-\varepsilon}\right)} = \frac{2\delta(5-3\varepsilon)}{1-\varepsilon-2\delta(1-\varepsilon)-2\delta}$. Therefore, if $\delta < 1/14$, then $\delta_s < \frac{2\delta(5-3\varepsilon)}{1-\varepsilon+2\delta\varepsilon-4\delta}\Big|_{\varepsilon=0} = \frac{10\delta}{1-4\delta} < 1$, as is necessary.

The last step is to show that C_A has the chance to commit before C_B has sent $n/2$ messages over the insertion-deletion channel, and vice versa. Recall that C_A (respectively C_B) commits when $\mathtt{I}_A$ (respectively $\mathtt{I}_B$) equals $n/2 - n\delta\left(1+\frac{1}{1-\varepsilon}\right)$. Let $\mathtt{i}_B$ be the number of messages sent by C_B over the insertion-deletion channel. The difference $|\mathtt{i}_B - \mathtt{I}_A|$ only increases due to an error, and a single error can only increase this difference by at most one. Therefore, when $\mathtt{i}_B = n/2$, $\mathtt{I}_A \geq n/2 - n\delta$, so C_A has already committed. Next, $\mathtt{I}_A$ only grows larger than $\mathtt{I}_B$ if there is an error or if C_B improperly decodes a synchronization symbol and erroneously chooses to not increase $\mathtt{I}_B$. Therefore, $\mathtt{I}_A$ is never more than $n\delta\left(1+\frac{1}{1-\varepsilon}\right)$ larger than $\mathtt{I}_B$. This means that when $\mathtt{I}_A = n/2$, it must be that $\mathtt{I}_B \geq n/2 - \left(1+\frac{1}{1-\varepsilon}\right)$, so C_B has committed. □

7.3.3 Binary interactive channel simulation

We now show that with the help of synchronization strings, a binary interactive insertion-deletion channel can be used to simulate a binary interactive substitution channel, inducing a $\widetilde{O}(\sqrt{\delta})$ fraction of bit-flips. In this way, the two communicating parties may interact as though they are communicating over a substitution channel. They therefore can employ substitution channel coding schemes while using the simulator as a black box means of converting the insertion-deletion channel to a substitution channel.

The key difference between this simulation and the one-way, large alphabet simulation is that Alice and Bob communicate through C_A and C_B for *blocks* of r rounds, between which C_A and C_B check if they are in sync. Due to errors, there may be times when Alice and Bob are in disagreement about which block, and what part of the block, they are in. C_A and C_B ensure that Alice and Bob are in sync most of the time.

When Alice sends C_A a message from a new block of communication, C_A holds that message and alerts C_B that a new block is beginning. C_A does this by sending C_B a header that is a string consisting of a single one followed by $s-1$ zeros (10^{s-1}). Then, C_A indicates which block Alice is about to start by sending a synchronization symbol to C_B. Meanwhile, when C_B receives a 10^{s-1} string, he listens for the synchronization symbol,

Algorithm 9 Simulation of a substitution channel using an insertion-deletion channel, at C_A's side

$\Pi \leftarrow n$-round interactive coding scheme over a substitution channel to be simulated
Initialize parameters: $r \leftarrow \sqrt{\frac{\log(1/\delta)}{\delta}}$; $R_{total} \leftarrow \left\lceil n\sqrt{\frac{\delta}{\log(1/\delta)}} \right\rceil$; $s \leftarrow c\log(1/\delta)$; $S \leftarrow \varepsilon$-synchronization string of length R_{total}
Reset Status: $\mathtt{i} \leftarrow 0$
for R_{total} iterations **do**
    Send s zeros to C_B
    Send $S[\mathtt{i}]$ to C_B
    For r rounds, relay messages between C_B and Alice
    $\mathtt{i} \leftarrow \mathtt{i} + 1$

makes his best guess about which block Alice is in, and then communicates with Bob and C_A accordingly. This might entail sending dummy blocks to Bob or C_A if he believes that they are in different blocks.Algorithms 9 and 10 detail C_A and C_B's protocol. To describe the guarantee that our simulation provides, we first define *block substitution channels.*

Definition 7.3.6 (Block Substitution Channel)**.** *An n-round adversarial substitution channel is called a* (δ, r)-block substitution channel *if the adversary is restricted to substitution $n\delta$ symbols which are covered by $n\delta/r$ blocks of r consecutively transmitted symbols.*

Theorem 7.3.7. *Suppose that n rounds of a binary interactive insertion-deletion channel with a δ fraction of insertions and deletions are given. For sufficiently small δ, it is possible to deterministically simulate $n(1 - \Theta(\sqrt{\delta\log(1/\delta)}))$ rounds of a binary interactive $(\Theta(\sqrt{\delta\log(1/\delta)}), \sqrt{(1/\delta)\log(1/\delta)})$-block substitution channel between two parties, Alice and Bob, assuming that all substrings of form 10^{s-1} where $s = c\log(1/\delta)$ that Alice sends can be covered by $n\delta$ intervals of $\sqrt{(1/\delta)\log(1/\delta)}$ consecutive rounds. The simulation is performed efficiently if the synchronization string is efficient.*

Proof. Suppose Alice and Bob communicate via intermediaries C_A and C_B who act according to Algorithms 9 and 10. In total, Alice and Bob will attempt to communicate n_s bits to one another over the simulated channel, while C_A and C_B communicate a total of n bits to one another. The adversary is allowed to insert or delete up to $n\delta$ symbols and C_A sends $n/2$ bits, so C_B may receive between $n/2 - n\delta$ and $n/2 + n\delta$ symbols. To prevent C_B from stalling indefinitely, C_B only listens to the first $n(1-2\delta)/2$ bits he receives.

For $r = \sqrt{(1/\delta)\log(1/\delta)}$, we define a *chunk* to be $r_c := (s + |\Sigma_{syn}| + r/2)$ consecutive bits that are sent by C_A to C_B. In particular, a chunk corresponds to a section header and synchronization symbol followed by $r/2$ rounds of messages sent from Alice. As C_B cares about the first $n(1-2\delta)/2$ bits it receives, there are $\frac{n(1-2\delta)}{2r_c}$ chunks in total. Hence, $n_s = \frac{n(1-2\delta)}{2r_c} \cdot r$ since C_B and C_A's communication is alternating.

Note that if Alice sends a substring of form 10^{s-1} in the information part of a chunk, then Bob mistakenly detects a new block. With this in mind, we say a chunk is *good* if:

Algorithm 10 Simulation of a substitution channel using an insertion-deletion channel, at C_B's side

$\Pi \leftarrow n$-round interactive coding scheme over a substitution channel to be simulated
Initialize parameters: $r \leftarrow \sqrt{\frac{\log(1/\delta)}{\delta}}$; $R_{total} \leftarrow \left\lfloor n(1-\delta)\sqrt{\frac{\delta}{\log(1/\delta)}} \right\rfloor$; $s \leftarrow c\log(1/\delta)$; $S \leftarrow \varepsilon$-synchronization string of length R_{total}
Reset Status: $\mathtt{i}, \mathtt{z}, \mathtt{I}_B \leftarrow 0$
for R_{total} iterations **do**
    while $\mathtt{z} < s$ **do**
        Receive b from C_A
        if $b = 0$ **then**
            $\mathtt{z} \leftarrow \mathtt{z} + 1$
        else
            $\mathtt{z} \leftarrow 0$
        Send dummy bit to C_A
    $\mathtt{z} \leftarrow 0$
    Receive m, the next $|\Sigma_{syn}|$ bits sent by C_A
    $\tilde{\mathtt{I}}_A \leftarrow$ Synchronization string decode(m, S)
    if $\tilde{\mathtt{I}}_A = \mathtt{I}_B$ **then**
        For r rounds, relay messages between C_A and Bob
        $\mathtt{I}_B \leftarrow \mathtt{I}_B + 1$
    if $\tilde{\mathtt{I}}_A < \mathtt{I}_B$ **then**
        For r rounds, send dummy messages to C_A
    if $\tilde{\mathtt{I}}_A > \mathtt{I}_B$ **then**
        For r rounds, send dummy messages to Bob
        For r rounds, relay messages between C_A and Bob
        $\mathtt{I}_B \leftarrow \mathtt{I}_B + 2$

1. There are no errors injected in the chunk or affecting C_B's detection of the chunk's header,

2. C_B correctly decodes the index that C_A sends during the chunk, and

3. C_A does not send a 10^{s-1} substring in the information portion of the chunk.

If a chunk is not good, we call it *bad*. If the chunk is bad because C_B does not decode C_A's index correctly even though they were in sync and no errors were injected, then we call it *decoding-bad*. If it is bad because Alice sends a 10^{s-1} substring, we call it *zero-bad* and otherwise, we call it *error-bad*. Throughout the protocol, C_B uses the variable $\mathtt{I}_B$ to denote the next index of the synchronization string C_B expects to receive and we use $\mathtt{I}_A$ to denote the index of the synchronization string C_A most recently sent. Notice that if a chunk is good and $\mathtt{I}_A = \mathtt{I}_B$, then all messages are correctly conveyed.

We now bound the maximum number of bad chunks that occur over the course of the simulation. Suppose the adversary injects errors into the i^{th} chunk, making that chunk bad. The $(i+1)^{th}$ chunk may also be bad, since Bob may not be listening for 10^{s-1} from C_A when C_A sends them, and therefore may miss the block header. However, if the adversary does not inject any errors into the $(i+1)^{th}$ and the $(i+2)^{th}$ chunk, then the $(i+2)^{th}$ chunk will be good. In effect, a single error may render at most two chunks useless. Since the adversary may inject $n\delta$ errors into the insertion-deletion channel, this means that the number of chunks that are error-bad is at most $2n\delta$. Additionally, by assumption, the number of zero-bad chunks is also at most $n\delta$.

We also must consider the fraction of rounds that are decoding-bad. In order to do this, we appeal to Theorem 7.2.5, which guarantees that if an ε-synchronization string of length N is sent over an insertion-deletion channel with a δ' fraction of insertions and deletions, then the receiver will decode the index of the received symbol correctly for all but $2N\delta'/(1-\varepsilon)$ symbols. In this context, N is the number of chunks, i.e. $N = n(1-2\delta)/(2r_c)$, and the fraction of chunks corrupted by errors is $\delta' = 4n\delta/N$. Therefore, the total number of bad chunks is at most $4\delta n + 2N\delta'/(1-\varepsilon) = 4\delta n(3-\varepsilon)/(1-\varepsilon)$.

We will now use these bounds on the number of good and bad chunks to calculate how many errors there are for Alice and Bob, communicating over the simulated channel. As noted, so long as the chunk is good and $\mathtt{I}_A = \mathtt{I}_B$, then all messages are correctly conveyed to Alice from Bob and vice versa. Meanwhile, a single error, be it an adversarial injection or a synchronization string improper decoding, may cause $|\mathtt{I}_B - \mathtt{I}_A|$ to increase by at most two since a single error may cause Bob to erroneously simulate Step 13 and therefore increment $\mathtt{I}_B$ by two when, in the worst case, $\mathtt{I}_A$ does not change. On the other hand, if $|\mathtt{I}_B - \mathtt{I}_A| \geq 1$ and the chunk is good, this difference will decrease by at least 1, as is clear from Lines 18 and 20 of Algorithm 10.

In total, we have that over the course of the computation, $|\mathtt{I}_B - \mathtt{I}_A|$ increases at most $\frac{3-\varepsilon}{1-\varepsilon} \cdot 4\delta n$ times and each time by at most 2. Therefore, there will be at most $\frac{3-\varepsilon}{1-\varepsilon} \cdot 8\delta n$ good chunks during which $|\mathtt{I}_B - \mathtt{I}_A| \geq 1$. This gives that all but $\frac{3-\varepsilon}{1-\varepsilon} \cdot 12\delta n$ chunks are good chunks and have $\mathtt{I}_A = \mathtt{I}_B$ upon their arrival on Bob's side. Remember that the total number of chunks is $\frac{n(1-2\delta)}{2r_c}$, hence, the simulated channel is a $\left(\frac{12\delta n(3-\varepsilon)/(1-\varepsilon)}{n(1-2\delta)/(2r_c)}, r\right) = \left(\frac{24\delta r_c(3-\varepsilon)}{(1-\varepsilon)(1-2\delta)}, r\right)$ block substitution channel.

Thus far, we have shown that one can simulate $n(1-2\delta)\frac{r}{2r_c}$ rounds of a $\left(\frac{24\delta r_c(3-\varepsilon)}{(1-\varepsilon)(1-2\delta)}, r\right)$-block substitution channel over a given channel as described in the theorem statement. More specifically, over $n(1-2\delta)$ rounds of communication over the insertion-deletion channel, a $\frac{2r_c-r}{2r_c} = \frac{s+|\Sigma_{syn}|}{r_c}$ fraction of rounds are used to add headers and synchronization symbols to chunks and a $\frac{24\delta r_c(3-\varepsilon)}{(1-\varepsilon)(1-2\delta)}$ fraction can be lost due to the errors injected by the adversary or 10^{s-1} strings in Alice's stream of bits. Therefore, the overall fraction of lost bits in this simulation is $\frac{s+|\Sigma_{syn}|}{r_c} + \frac{24\delta r_c(3-\varepsilon)}{(1-\varepsilon)(1-2\delta)}$. Since $s = c\log\frac{1}{\delta}$, ε and $|\Sigma_{syn}|$ are constants, and $1-2\delta$ approaches to one for small δ, the optimal asymptotic choice is $r = \sqrt{(1/\delta)\log 1/\delta}$. This choice gives a simulated channel with characteristics described in the theorem statement. This simulation is performed efficiently if the synchronization symbols are efficiently computable. □

The simulation stated in Theorem 7.3.7 burdens an additional condition on Alice's stream of bits by requiring it to have a limited number of substrings of form 10^{s-1}. We now introduce a high probability technique to modify a general interactive communication protocol in a way that makes all substrings of form 10^{s-1} in Alice's stream of bits fit into $n\delta$ intervals of length $r = \sqrt{(1/\delta)\log(1/\delta)}$.

Lemma 7.3.8. *Assume that n rounds of a binary interactive insertion-deletion channel with an oblivious adversary who is allowed to inject $n\delta$ errors are given. There is a pre-coding scheme that can be utilized on top of the simulation introduced in Theorem 7.3.7. It modifies the stream of bits sent by Alice so that with probability $1 - e^{-\frac{c-3}{2}n\delta\log\frac{1}{\delta}(1+o(1))}$, all substrings of form 10^{s-1} where $s = c\log(1/\delta)$ in the stream of bits Alice sends over the simulated channel can be covered by $n\delta$ intervals of length $r = \sqrt{(1/\delta)\log(1/\delta)}$. This pre-coding scheme comes at the cost of a $\Theta(\sqrt{\delta\log(1/\delta)})$ fraction of the bits Alice sends through the simulated channel.*

Proof. Note that in the simulation process, each $\frac{r}{2}$ consecutive bits Alice sends will form one of the chunks C_A sends to C_B alongside some headers. The idea of this pre-coding is simple. Alice uses the first $\frac{s}{2}$ data bits (and not the header) of each chunk to share $\frac{s}{2}$ randomly generated bits with Bob (instead of running the interactive protocol) and then both of them extract a string S' of $\frac{r}{2}$ $\frac{s}{2}$-wise independent random variables. Then, Alice XORs the rest of data bits she passes to C_A with S' and Bob XORs those bits with S' again to retrieve the original data.

We now determine the probability that the data block of a chunk of the simulation contains a substring of form 10^{s-1}. Note that if a block of size $\frac{r}{2}$ contains a 10^{s-1} substring, then one of its substrings of length $\frac{s}{2}$ starting at positions $0, \frac{s}{2}, \frac{2s}{2}, \cdots$ is all zero. Since P is $\frac{s}{2}$-wise independent, the probability of each of these $\frac{s}{2}$ substrings containing only zeros is $2^{-\frac{s}{2}} = \delta^{\frac{c}{2}}$. Taking a union bound over all these substrings, the probability of a block containing a 10^{s-1} substring can be bounded above by

$$p = \frac{r/2}{s/2}\cdot\delta^{\frac{c}{2}} = \sqrt{\frac{\delta^{c-1}}{\log(1/\delta)}}.$$

Now, we have:

$$\begin{aligned}\Pr\left\{\text{Number of blocks containing } 10^{s-1} > n\delta\right\} &< \binom{\frac{n}{s}}{n\delta}p^{n\delta} \le \left(\frac{ne}{ns\delta}\right)^{n\delta}\left(\sqrt{\frac{\delta^{c-1}}{\log(1/\delta)}}\right)^{n\delta}\\ &< \left(\frac{\delta^{(c-3)/2}e}{c\log^{3/2}(1/\delta)}\right)^{n\delta} = e^{-\frac{c-3}{2}n\delta\log\frac{1}{\delta}(1+o(1))}.\end{aligned}$$

□

Applying this pre-coding for $c \ge 3$ on top of the simulation from Theorem 7.3.7 implies the following.

Theorem 7.3.9. *Suppose that n rounds of a binary interactive insertion-deletion channel with a δ fraction of insertions and deletions performed by an oblivious adversary are given. For sufficiently small δ, it is possible to simulate $n(1-\Theta(\sqrt{\delta\log(1/\delta)}))$ rounds of a binary interactive $(\Theta(\sqrt{\delta\log(1/\delta)}), \sqrt{(1/\delta)\log 1/\delta})$-block substitution channel between two parties over the given channel. The simulation works with probability $1-\exp(-\Theta(n\delta\log(1/\delta)))$ and is efficient if the synchronization string is efficient.*

Lemma 7.3.10. *Suppose that n rounds of a binary, interactive, fully adversarial insertion-deletion channel with a δ fraction of insertions and deletions are given. The pre-coding scheme proposed in Lemma 7.3.8 ensures that the stream of bits sent by Alice contains fewer than $n\delta$ substrings of form 10^{s-1} for $s = c\log(1/\delta)$ and $c > 5$ with probability $1-e^{-\Theta(n\delta\log(1/\delta))}$.*

Proof. Lemma 7.3.8 ensures that for a fixed adversarial error pattern, the stream of bits sent by Alice contains fewer than $n\delta$ substrings of form 10^{s-1} upon applying the pre-coding scheme. However, in the fully adversarial setting, the adversary need not fix the error pattern in advance. Since the communication is interactive, the adversary can thus adaptively alter the bits Alice chooses to send. In this proof, we take a union bound over all error patterns with a δ fraction of errors and show that with high probability, upon applying the pre-coding scheme, the stream of bits sent by Alice contains fewer than $\Theta(n\delta)$ substrings of form 10^{s-1}.

We claim that the number of error patterns with exactly k insertions or deletions is at most $3^k\binom{n+k}{k}$. Note that if symbols $s_1,\ldots,s_n$ are being sent, each of the k errors can potentially occur within the intervals $[s_1,s_2),[s_2,s_3),\ldots,[s_{n-1},s_n)$, or after s_n is sent. Each error could be a deletion, insertion of "1", or insertion of "0". This gives the claimed error pattern count. Further, any error pattern with fewer than $k-1$ errors can be thought of as an error pattern with either $k-1$ or k errors where the adversary deletes an arbitrary set of symbols and then inserts the exact same symbols immediately. Therefore, the number of all possible error patterns with at most $n\delta$ insertions or deletions can be upper-bounded by

$$\sum_{k=n\delta-1}^{n\delta} 3^k\binom{n+k}{k} \le 2\cdot 3^{n\delta}\binom{n(1+\delta)}{n\delta} \le 2\cdot 3^{n\delta}\left(\frac{n(1+\delta)e}{n\delta}\right)^{n\delta} < 2\left(\frac{6e}{\delta}\right)^{n\delta} = e^{n\delta\log\frac{1}{\delta}(1+o(1))}.$$

Now, since summation of $\exp\left(n\delta\log(1/\delta)(1+o(1))\right)$ many probabilities any of which smaller than $\exp\left(-\frac{c-3}{2}n\delta\log(1/\delta)(1+o(1))\right)$ is still $\exp\left(-\Theta\left(n\delta\log(1/\delta)\right)\right)$ for $c>5$, the probability of this pre-coding making more than $n\delta$ disjoint 10^{s-1} substrings for $s = c\log\frac{1}{\delta}$ in fully adversarial setting is again $1-\exp\left(-\Theta\left(n\delta\log(1/\delta)\right)\right)$. □

Theorem 7.3.7 and Lemma 7.3.10 allow us to conclude that one can perform the simulation stated in Theorem 7.3.7 over any interactive protocol with high probability (see Theorem 7.3.11).

Theorem 7.3.11. *Suppose that n rounds of a binary interactive insertion-deletion channel with a δ fraction of insertions and deletions performed by a non-oblivious adversary*

are given. For a sufficiently small δ, it is possible to simulate $n\left(1-\Theta\left(\sqrt{\delta\log(1/\delta)}\right)\right)$ rounds of a binary interactive $\left(\Theta\left(\sqrt{\delta\log(1/\delta)}\right),\sqrt{\frac{\log 1/\delta}{\delta}}\right)$-block substitution channel between two parties, Alice and Bob, over the given channel. The simulation is efficient if the synchronization string is efficient and works with probability $1-\exp\left(-\Theta\left(n\delta\log(1/\delta)\right)\right)$.

7.3.4 Binary One Way Communication

It is trivial to simplify Algorithms 9 and 10 from Section 7.3 to prove our simulation guarantees over binary alphabets. Specifically, the messages that Bob sends may be completely ignored and thereby we immediately obtain the following result for one-way insertion-deletion channels:

Theorem 7.3.12. *Suppose that n rounds of a binary one-way insertion-deletion channel with a δ fraction of insertions and deletions are given. For a sufficiently small δ, it is possible to deterministically simulate $n\left(1-\Theta\left(\sqrt{\delta\log(1/\delta)}\right)\right)$ rounds of a binary*

$$\left(\Theta\left(\sqrt{\delta\log\frac{1}{\delta}}\right),\sqrt{\frac{\log(1/\delta)}{\delta}}\right)$$

one-way block substitution channel between two parties, Alice and Bob, over the given channel assuming that all substrings of form 10^{s-1} for $s=c\log\frac{1}{\delta}$ in Alice's stream of bits can be covered by $n\delta$ intervals of $\sqrt{\frac{\log(1/\delta)}{\delta}}$ consecutive rounds. This simulation is performed efficiently if the synchronization string is efficient.

Further, one can use the pre-coding technique introduced in Lemma 7.3.8 and Theorem 7.3.12 to show that:

Theorem 7.3.13. *Suppose that n rounds of a binary one-way insertion-deletion channel with a δ fraction of insertions and deletions are given. For sufficiently small δ, it is possible to simulate $n\left(1-\Theta\left(\sqrt{\delta\log(1/\delta)}\right)\right)$ rounds of a binary $\left(\Theta\left(\sqrt{\delta\log(1/\delta)}\right),\sqrt{\frac{\log 1/\delta}{\delta}}\right)$ block substitution channel between two parties, Alice and Bob, over the given channel. The simulation works with probability $1-e^{-\Theta(n\delta\log(1/\delta))}$, and is efficient if the synchronization string is efficient.*

7.4 Applications: Binary Insertion-Deletion Codes

The binary one-way simulation in Theorem 7.3.13 suggests a natural way to overcome insertion-deletion errors in one-way binary channels. One can simply simulate the substitution channel and use appropriate substitution channel error correcting codes on top of the simulated channel to make the communication resilient to insertion-deletion errors. However, as the simulation works with high probability, this scheme is not deterministic.

As preserving the streaming quality is not necessary for the sake of designing binary insertion-deletion codes, we can design a deterministic pre-coding and error correcting code that can be used along with the deterministic simulation introduced in Theorem 7.3.12 to generate binary insertion-deletion codes.

Lemma 7.4.1. *There exist error correcting codes for* $\left(\Theta\left(\sqrt{\delta \log(1/\delta)}\right), \sqrt{\log(1/\delta)/\delta}\right)$ *block substitution channels with rate* $1 - \Theta(\sqrt{\delta \log(1/\delta)}) - \delta^{c/2}$ *whose codewords are guaranteed to be free of substrings of form* 10^{s-1} *for* $c \log \frac{1}{\delta}$.

Proof. Assume that we are sending n bits over a

$$(p_b, r_b) = \left(\Theta\left(\sqrt{\delta \log \frac{1}{\delta}}\right), \sqrt{\frac{\log(1/\delta)}{\delta}}\right)$$

block substitution channel. Let us chop the stream of bits into blocks of size r_b. Clearly, the fraction of blocks which may contain any errors is at most $2p_b$. Therefore, by looking at each block as a symbol from a large alphabet of size 2^{r_b}, one can protect them against $2p_b$ fraction of errors by having $\Theta(H_{2^{r_b}}(2p_b) + p_b)$ fraction of redundant blocks.

In the next step, we propose a way to efficiently transform each block of length r_b of the encoded string into a block of $r_b(1 + \delta^{c/2})$ bits so that the resulting stream of bits be guaranteed not to contain 10^{s-1} substrings.

To do so, we think of each block of length r_b as a number in the range of zero to $2^{r_b} - 1$. Then, we can represent this number in $2^{s/2} - 1$ base and then map each of the symbols of this presentation to strings of $\frac{s}{2}$ bits except the $\frac{s}{2}$ all-zero string. This way, one can efficiently code the string into a stream of bits free of 10^{s-1} substrings by losing a $2^{-s/2}$-fraction of bits.

On the other side of the channel, Bob has to split the stream he receives into blocks of length $\frac{s}{2}$, undo the map to find out a possibly corrupted version of the originally encoded message and then decode the 2^{r_b}-sized alphabet error correcting code to extract Alice's message.

This introduces an insertion-deletion code with rate of $1 - \Theta(H_{2^{r_b}}(2p_b) + p_b) - 2^{-s/2}$. As $s = c \log \frac{1}{\delta}$, $2^{-s/2} = \delta^{c/2}$, $p_b = \sqrt{\delta \log \frac{1}{\delta}}$, and

$$H_{2^{r_b}}(2p_b) = \Theta(2p_b \log_{2^{r_b}}(1/2p_b)) = \Theta\left(\frac{2p_b \log_2(1/2p_b)}{r_b}\right) = \Theta\left(\frac{\sqrt{\delta \log \frac{1}{\delta}} \log \frac{1}{\delta}}{\sqrt{\frac{\log(1/\delta)}{\delta}}}\right) = \Theta\left(\delta \log \frac{1}{\delta}\right)$$

the proof is complete. □

One can set $c \geq 1$ and then apply such error correcting codes on top of a simulated channel as described in Theorem 7.3.12 to construct a binary insertion-deletion code resilient to δ fraction of insertions and deletions with rate $1 - \Theta\left(\sqrt{\delta \log \frac{1}{\delta}}\right)$ for a sufficiently small δ.

Theorem 7.4.2. *There exists a constant $0 < \delta_0 < 1$ such that for any $0 < \delta < \delta_0$ there is a binary insertion-deletion code with rate $1 - \Theta\left(\sqrt{\delta \log \frac{1}{\delta}}\right)$ which is decodable from δ fraction of insertions and deletions.*

7.5 Applications: New Interactive Coding Schemes

Efficient Coding Scheme Tolerating $1/44$ Fraction of Errors. In this section, we will provide an efficient coding scheme for interactive communication over insertion-deletion channels by first making use of large alphabet interactive channel simulation provided in Theorem 7.3.5 to effectively transform the given channel into a simple substitution interactive channel and then use the efficient constant-rate coding scheme of Ghaffari and Haeupler [GH14] on top of the simulated channel. This will give an efficient constant-rate interactive communication over large enough constant alphabets as described in Theorem 7.1.2. We review the following theorem of Ghaffari and Haeupler [GH14] before proving Theorem 7.1.2.

Theorem 7.5.1 (Theorem 1.1 from [GH14])**.** *For any constant $\varepsilon > 0$ and n-round protocol Π there is a randomized non-adaptive coding scheme that robustly simulates Π against an adversarial error rate of $\rho \leq 1/4 - \varepsilon$ using $N = O(n)$ rounds, a near-linear $n \log^{O(1)} n$ computational complexity, and failure probability $2^{-\Theta(n)}$.*

Proof of Theorem 7.1.2. For a given insertion-deletion interactive channel over alphabet Σ suffering from δ fraction of edit-corruption errors, Theorem 7.3.5 enables us to simulate $n - 2n\delta(1 + (1-\varepsilon')^{-1})$ rounds of ordinary interactive channel with $\frac{2\delta(5-3\varepsilon')}{1-\varepsilon'+2\varepsilon'\delta-4\delta}$ fraction of symbol by designating $\log |\Sigma_{syn}|$ bits of each symbol to index simulated channel's symbols with an ε'-synchronization string over Σ_{syn}.

One can employ the scheme of Ghaffari and Haeupler [GH14] over the simulated channel as long as error fraction is smaller than $1/4$. Note that $\frac{2\delta(5-3\varepsilon')}{1-\varepsilon'+2\delta\varepsilon'-4\delta}\Big|_{\varepsilon'=0} = \frac{10\delta}{1-4\delta} < \frac{1}{4} \Leftrightarrow \delta < \frac{1}{44}$. Hence, as long as $\delta = 1/44 - \varepsilon$ for $\varepsilon > 0$, for small enough $\varepsilon' = O_\varepsilon(1)$, the simulated channel has an error fraction that is smaller than $1/4$. Therefore, by running the efficient coding scheme of Theorem 7.5.1 over this simulated channel one gets a constant rate coding scheme for interactive communication that is robust against $1/44 - \varepsilon$ fraction of edit-corruptions. Note that this simulation requires the alphabet size to be large enough to contain synchronization symbols (which can come from a polynomially large alphabet in terms of ε') and also meet the alphabet size requirements of Theorem 7.5.1. This requires the alphabet size to be $\Omega_\varepsilon(1)$, i.e., a large enough constant merely depending on ε. The success probability and time complexity are direct consequences of Theorem 7.5.1 and Theorem 7.2.5. □

Efficient Coding Scheme with Near-Optimal Rate over Small Alphabets. In this section we present another insertion-deletion interactive coding scheme that achieves near-optimal communication efficiency as well as computation efficiency by employing a similar idea as in Section 7.5.

In order to derive a rate-efficient interactive communication coding scheme over small alphabet insertion-deletion channels, Algorithms 9 and 10 can be used to simulate a substitution channel and then the rate-efficient interactive coding scheme for substitution channels introduced by Haeupler [Hae14] can be used on top of the simulated channel.

We start by a quick review of Theorems 7.1 and 7.2 of Haeupler [Hae14].

Theorem 7.5.2 (Theorem 7.1 from [Hae14])**.** *Suppose any n-round protocol Π using any alphabet Σ. Algorithm 3 [from [Hae14]] is a computationally efficient randomized coding scheme which given Π, with probability $1-2^{-\Theta(n\delta)}$, robustly simulates it over any oblivious adversarial error channel with alphabet Σ and error rate δ. The simulation uses $n(1+\Theta(\sqrt{\delta}))$ rounds and therefore achieves a communication rate of $1-\Theta(\sqrt{\delta})$.*

Theorem 7.5.3 (Theorem 7.2 from [Hae14])**.** *Suppose any n-round protocol Π using any alphabet Σ. Algorithm 4 [from [Hae14]] is a computationally efficient randomized coding scheme which given Π, with probability $1-2^{-\Theta(n\delta)}$, robustly simulates it over any fully adversarial error channel with alphabet Σ and error rate δ. The simulation uses $n(1+\Theta\left(\sqrt{\delta\log\log\frac{1}{\delta}}\right)$ rounds and therefore achieves a communication rate of $1-\Theta\left(\sqrt{\delta\log\log\frac{1}{\delta}}\right)$.*

The interaction between the error rate and rate loss provided in Theorems 7.5.2 and 7.5.3 (Theorems 7.1 and 7.2 of [Hae14]) leads us to the following corollary.

Corollary 7.5.4. *There are high-probability efficient coding schemes for interactive communication over insertion-deletion channels that are robust against δ fraction of edit-corruptions for sufficiently small δ and have communication rate of $1-\Theta\left(\sqrt[4]{\delta\log\frac{1}{\delta}}\right)$ against oblivious adversaries and $1-\Theta\left(\sqrt[4]{\delta\log\frac{1}{\delta}\log^2\log\frac{1}{\delta}}\right)$ in fully adversarial setup.*

However, these results can be improved upon by taking a closer look at the specifics of the interactive communication coding scheme in [Hae14].

In a nutshell, the interactive coding scheme proposed in [Hae14] simulates an interactive protocol Π by splitting the communication into iterations. In each iteration, the coding scheme lets Alice and Bob communicate for a block of $\bar{r}$ rounds, then uses r_c rounds of communication after each block so the parties can verify if they are on the same page and then decide whether to continue the communication or to roll back the communication for some number of iterations. The parties perform this verification by exchanging a fingerprint (hash) of their versions of the transcript. Next, each party checks if the fingerprint he receives matches his own, which in turn identifies whether the parties agree or disagree about the communication transcript. Based on the result of this check, each party decides if he should continue the communication from the point he is already at or if he should roll back the communication for some number of iterations and continue from there. (see Algorithms 3 and 4 of [Hae14])

The analysis in Section 7 of Heaupler [Hae14] introduces a potential function Φ which increases by at least one whenever a round of communication is free of any errors or hash collisions. A *hash collision* occurs if the parties' transcripts do not agree due to errors

that occurred previously, yet the two parties' fingerprints erroneously match. The analysis also shows that whenever a round of communication contains errors or hash collisions, regardless of the number of errors or hash collisions happening in a round, the potential drops by at most a fixed constant. (Lemmas 7.3 and 7.4 of [Hae14])

For an error rate of $\bar{\delta}$, there can be at most $n\bar{\delta}$ rounds suffering from an error. Haeupler [Hae14] shows that the number of hash collisions can also be bounded by $\Theta(n\bar{\delta})$ with exponentially high probability. (Lemmas 7.6 and 7.7 and Corollary 7.8 of [Hae14]) Given that the number of errors and hash collisions is bounded by $\Theta(n\bar{\delta})$, Haeupler [Hae14] shows that if $\Phi > \frac{n}{\bar{r}} + \Theta(n\bar{\delta})$, then the two parties will agree on the first n steps of communication and therefore the communication will be simulated thoroughly and correctly. Therefore, after $\frac{n}{\bar{r}} + \Theta(n\bar{\delta})$ rounds the simulation is complete, and the rate of this simulation is $1 - \Theta\left(r\bar{\delta} + \frac{r_c}{\bar{r}}\right)$.

Theorem 7.5.5 (Interactive Coding against Block Substitution)**.** *By choosing an appropriate block length in the Haeupler [Hae14] coding scheme for oblivious adversaries (Theorem 7.5.2), one obtains a robust efficient interactive coding scheme for (δ_b, r_b)-block substitution channel with communication rate $1 - \Theta(\sqrt{\delta_b \max\{\delta_b, 1/r_b\}})$ that works with probability $1 - 2^{-\Theta(n\delta_b/r_b)}$.*

Proof. Let us run Haeupler [Hae14] scheme with block size $\bar{r}$. This way, each block of substitution that rises in channel, may corrupt $\max\left\{\frac{r_b}{\bar{r}}, 1\right\}$ blocks of Haeupler [Hae14] scheme. Therefore, the total number of corrupted blocks in Haeupler [Hae14] scheme can be:

$$\frac{n\delta_b}{r_b} \max\left\{\frac{r_b}{\bar{r}}, 1\right\}$$

Therefore, the total fraction of Haeupler [Hae14] scheme's blocks containing errors is at most

$$\bar{\delta} = \delta_b \max\left\{1, \frac{\bar{r}}{r_b}\right\}$$

For oblivious adversaries, Haeupler [Hae14] suggests a verification process which can be performed in $r_c = \Theta(1)$ steps at the end of each round. We use the exact same procedure. Lemma 7.6 of [Hae14] directly guarantees that the fraction of hash collisions using this procedure is upper-bounded by $\Theta(\bar{\delta})$ with probability $1 - 2^{-\Theta(n\bar{\delta}/\bar{r})}$. As the fraction of blocks suffering from hash collisions or errors is at most $\Theta(\bar{\delta})$, the communication can be shown to be complete in $(1 + \bar{\delta})$ multiplicative factor of rounds by the same potential argument as in [Hae14].

Therefore, the rate lost in this interactive coding scheme is

$$\Theta\left(\bar{\delta} + \frac{r_c}{\bar{r}}\right) = \Theta\left(\delta_b \max\left\{1, \frac{\bar{r}}{r_b}\right\} + \frac{1}{\bar{r}}\right)$$

Now, the only remaining task is to find $\bar{r}$ that minimizes the rate loss mentioned above. If we choose $\bar{r} \leq r_b$, the best choice is $\bar{r} = r_b$ as it reduces the rate to $\delta_b + \frac{1}{r_b}$. On the other hand if we choose $\bar{r} \geq r_b$, the optimal choice is $\bar{r} = \sqrt{\frac{r_b}{\delta_b}}$ if $\sqrt{\frac{r_b}{\delta_b}} \geq r_b \Leftrightarrow r_b \leq \frac{1}{\delta_b}$ or $\bar{r} = r_b$

otherwise. Hence, we set

$$\bar{r} = \begin{cases} \sqrt{\frac{r_b}{\delta_b}} & \text{if } r_b \leq \frac{1}{\delta_b} \\ r_b & r_b > \frac{1}{\delta_b} \end{cases}$$

Plugging this values for $\bar{r}$ gives that:

$$Rate = \begin{cases} 1 - \Theta\left(\sqrt{\frac{\delta_b}{r_b}}\right) & r_b \leq \frac{1}{\delta_b} \\ 1 - \Theta\left(\delta_b\right) & r_b > \frac{1}{\delta_b} \end{cases} = 1 - \Theta\left(\sqrt{\delta_b \max\left\{\delta_b, \frac{1}{r_b}\right\}}\right)$$

Also, the probability of this coding working correctly is:

$$1 - 2^{-\Theta(n\bar{\delta}/\bar{r})} = 1 - 2^{-\delta_b \max\left\{\frac{1}{\bar{r}}, \frac{1}{r_b}\right\}} = \begin{cases} 1 - 2^{-\delta_b \max\left\{\sqrt{\frac{\delta_b}{r_b}}, \frac{1}{r_b}\right\}} & r_b \leq \frac{1}{\delta_b} \\ 1 - 2^{-\delta_b \max\left\{\frac{1}{r_b}, \frac{1}{r_b}\right\}} & r_b > \frac{1}{\delta_b} \end{cases} = 1 - 2^{-\Theta(\frac{\delta_b}{r_b} n)}$$

□

Applying the coding scheme of Theorem 7.5.5 over the simulation from Theorem 7.3.9 implies the following.

Theorem 7.5.6. *For sufficiently small δ, there is an efficient interactive coding scheme over binary insertion-deletion channels which, is robust against δ fraction of edit-corruptions by an oblivious adversary, achieves a communication rate of $1 - \Theta(\sqrt{\delta \log(1/\delta)})$, and works with probability $1 - 2^{-\Theta(n\delta)}$.*

Moreover, we show that this result is extendable for the fully adversarial setup, as summarized in Theorem 7.1.3.

Proof of Theorem 7.1.3. Similar to the proofs of Theorems 7.5.5 and 7.5.6, we use the simulation discussed in Theorem 7.3.11 and coding structure introduced in Haeupler [Hae14] with rounds of length $\bar{r} = \sqrt{\frac{\log(1/\delta)}{\delta}}$ on top of that. However, this time, we use another verification strategy with length of $r_c = \log\log\frac{1}{\delta}$ which is used in Haeupler [Hae14] as a verification procedure in the interactive coding scheme for fully adversarial channels.

The idea of this proof, similar to the proof of Theorem 7.5.5, is to show that the number of rounds suffering from errors or hash collisions can be bounded by $\Theta(n\delta)$ with high probability and then apply the same potential function argument. All of the steps of this proof are, like their analogs in Theorem 7.5.5, implications of Haeupler[Hae14], except for the fact that the number of hash collisions can be bounded by $\Theta(n\delta)$. This is because of the fact that the entire Haeupler [Hae14] analysis, except Lemma 7.7 that bounds hash collisions, are merely based on the fact that all but $O(n\delta)$ rounds are error free.

Therefore, if we show that the number of hash collisions in the fully adversarial case is bounded by $\Theta(n\delta)$, the simulation rate will be

$$1 - \left(\frac{\Theta(n\delta)}{n/r} + \frac{r_c}{\bar{r}}\right) = 1 - \Theta\left(\sqrt{\frac{\log(1/\delta)}{\delta}} \cdot \delta + \frac{\log\log\frac{1}{\delta}}{\sqrt{\frac{\log(1/\delta)}{\delta}}}\right) = 1 - \Theta\left(\sqrt{\delta \log\frac{1}{\delta}}\right)$$

and the proof will be complete.

We now bound the number of hash collisions in our interactive coding scheme. The verification process for fully adversarial setting uses a two level hash function $hash_2(hash_1(\cdot))$, where the seed of $hash_1$ is randomly generated in each round.

Lemma 7.7 of Haeupler [Hae14] implies that for any oblivious adversary, the number of hash collisions due to $hash_1$ is at most $\Theta(n\delta)$ with probability $1 - \delta^{\Theta(n\delta)}$. To find a similar bound for non-oblivious adversaries, we count the number of all possible oblivious adversaries and then use a union bound. The number of oblivious adversaries is shown to be less than $\left(\frac{3e}{\delta}\right)^{n\delta} = e^{n\delta \log(1/\delta)(1+o(1))}$ in Lemma 7.3.10. Hence, the probability of having more than $\Theta(n\delta)$ $hash_1$ hash collisions in any fully adversarial scenario is at most $2^{-\Theta(n\delta)}$. Now, Corollary 7.8 of Haeupler [Hae14] can be directly applied to show that the number of $hash_2$ hash collisions can also be bounded by $\Theta(n\delta)$ with probability $1 - 2^{-\Theta(n\delta)}$. This completes the proof. □

This insertion-deletion interactive coding scheme is, to the best of our knowledge, the first to be computationally efficient, to have communication rate approaching one, and to work over arbitrarily small alphabets.

7.6 Synchronization Strings and Edit-Distance Tree Codes

We start by providing a new upper bound on the error tolerance of Braverman et al.'s coding scheme for interactive communication with a large alphabet over a insertion-deletion channel [BGMO17]. We tweak the definition of edit-distance tree codes, the primary tool that Braverman et al. use in their coding scheme. In doing so, we show that their scheme has an error tolerance of $1/10 - \varepsilon$ rather than $1/18 - \varepsilon$, which is the upper bound provided by Braverman et al. In particular, we prove the following theorem, which is a restatement of Theorem 1.4 from Braverman et al.'s work except for the revised error tolerance. The proof can be found in Section 7.6.1.

Theorem 7.6.1. *For any $\varepsilon > 0$, and for any binary (noiseless) protocol Π with communication $CC(\Pi)$, there exists a noise-resilient coding scheme with communication $O_\varepsilon(CC(\Pi))$ that succeeds in simulating Π as long as the adversarial edit-corruption rate is at most $1/10 - \varepsilon$.*

We then review the definition and key characteristics of edit-distance tree codes and discuss the close relationship between edit-distance tree codes and synchronization strings using the revised definition of edit-distance tree codes.

7.6.1 Revised definition of edit-distance tree codes

Before proceeding to the definition of edit-distance tree codes, we begin with some definitions.

Definition 7.6.2 (Prefix Code, Definition 3.1 of [BGMO17])**.** *A prefix code $C : \Sigma_{in}^n \to \Sigma_{out}^n$ is a code such that $C(x)[i]$ only depends on $x[1, i]$. C can also be considered as a $|\Sigma_{in}|$-ary tree of depth n with symbols written on edges of the tree using the alphabet Σ_{out}. On this tree, each tree path from the root of length l corresponds to a string $x \in \Sigma_{out}^l$ and the symbol written on the deepest edge of this path corresponds to $C(x)[l]$.*

Definition 7.6.3 (ε-bad Lambda, Revision of Definition 3.2 of [BGMO17])**.** *We say that a prefix code C contains an ε-bad lambda if when this prefix code is represented as a tree, there exist four tree nodes A, B, D, E such that $B \neq D$, $B \neq E$, B is D and E's common ancestor, A is B's ancestor or B itself, and $ED(AD, BE) \leq (1-\varepsilon) \cdot (|AD| + |BE|)$.*[1] *Here AD and BE are strings of symbols along the tree path from A to D and the tree path from B to E. See Figure 7.2 for an illustration.*

Definition 7.6.4 (Edit-distance Tree Code, Definition 3.3 of [BGMO17])**.** *We say that a prefix code $C : \Sigma_{in}^n \to \Sigma_{out}^n$ is an ε-edit-distance tree code if C does not contain an ε-bad lambda.*

Using this revised definition of edit-distance tree code, we prove Theorem 7.6.1 as follows:

Proof of Theorem 7.6.1. We relax Braverman et al.'s definition of an *edit-distance tree code* [BGMO17], which is an adaptation of Schulman's original tree codes [Sch93]. Edit-distance tree codes are parameterized by a real value ε, and Braverman et al. define an ε-edit-distance tree code to be a prefix code $C : \Sigma_{in}^n \to \Sigma_{out}^n$ that does not contain what they refer to as an *ε-bad lambda.* The code C contains an ε-bad lambda if when one considers C as a tree, there exist four tree nodes A, B, D, E such that $B \neq D$, $B \neq E$, B is D and E's common ancestor, A is B's ancestor or B itself, and $ED(AD, BE) \leq (1 - \varepsilon) \cdot \max(|AD|, |BE|)$. Here AD and BE are strings of symbols along the tree path from A to D and the tree path from B to E. See Figure 7.2 for an illustration. We relax the definition of an ε-bad lambda to be any four tree nodes A, B, D, E as above such that $ED(AD, BE) \leq (1 - \varepsilon) \cdot (|AD| + |BE|)$.

We use synchronization strings together with Schulman's original tree codes to prove that edit-distance tree codes, under this revised definition, exist. In particular, in Theorem 7.6.9, we show that synchronization strings can be concatenated with tree codes to produce edit-distance tree codes. Given that tree codes and synchronization strings exist, this means that edit-distance tree codes according to our revised definition exist.

As we saw in Lemma 7.6.6, if $C : \Sigma_{in}^n \to \Sigma_{out}^n$ is an ε-edit-distance tree code and $\tilde{c} \in \Sigma_{out}^m$, then there exists at most one $c \in \cup_{i=1}^n C(\Sigma_{in}^n)[1, i]$ such that $SD(c, \tilde{c}) \leq 1 - \varepsilon$. This leads to an improved version of Lemma 4.2 from Braverman et al.'s work, which we describe after we set up notation. In Braverman et al.'s protocol, let N_A and N_B be the number of messages Alice and Bob have sent when one of them reaches the end of the protocol. Let s_A and r_A be the messages Alice has sent and received over the course of the protocol, and let s_B and r_B be the messages Bob has sent and received over the course

[1] Braverman et al. say that A, B, D, E form an ε-bad lambda if $ED(AD, BE) \leq (1 - \varepsilon) \cdot \max(|AD|, |BE|)$.

of the protocol. Let $\tau_A = (\tau_1, \tau_2)$ be the string matching between $s_B[1, N_B]$ and $r_A[1, N_A]$ and let $\tau_B = (\tau_3, \tau_4)$ be the string matching between $s_A[1, N_A]$ and $r_B[1, N_B]$. Braverman et al. call a given round a *good decoding* if Alice correctly decodes the entire set of tree code edges sent by Bob. Lemma 7.6.6 admits the following improved version of Lemma 4.2 from Braverman et al's work:

Lemma 7.6.5 (Revision of Lemma 4.2 from [BGMO17])**.** *Alice has at least* $N_A + \left(1 - \frac{1}{1-\varepsilon}\right) sc(\tau_2) - \left(1 + \frac{1}{1-\varepsilon}\right) sc(\tau_1)$ *good decodings. Bob has at least* $N_B + \left(1 - \frac{1}{1-\varepsilon}\right) sc(\tau_4) - \left(1 + \frac{1}{1-\varepsilon}\right) sc(\tau_3)$ *good decodings.*

The proof of this revised lemma follows the exact same logic as the proof of Lemma 4.2 in Braverman et al.'s work. The theorem statement now follows the same logic as the proof of Theorem 4.4 in Braverman et al.'s work. We include the revised section of the proof below, where the only changes are the maximum error tolerance ρ, the value of n (which is smaller by a factor of $\frac{1}{2}$), and several equations. We set $\rho = \frac{1}{10} - \varepsilon$ and $n = \lceil \frac{T}{8\varepsilon} \rceil$, where T is the length of the original protocol. Let g_A be the number of good decodings of Alice and $b_A = N_A - g_A$ be the number of bad decodings of Alice. Similarly, let g_B be the number of good decodings of Bob and $b_B = N_B - g_B$ be the number of bad decodings of Bob. In Braverman et al.'s protocol, Alice and Bob share an edit-distance tree code $C : \Sigma_{in}^n \to \Sigma_{out}^n$. By definition of edit distance, $sc(\tau_1) + sc(\tau_3) = sc(\tau_2) + sc(\tau_4) \leq 2\rho n$. By Lemma 7.6.5,

$$\begin{aligned} b_A + b_B &\leq \frac{1}{1-\varepsilon}(sc(\tau_1) + sc(\tau_2)) + sc(\tau_1) - sc(\tau_2) + \frac{1}{1-\varepsilon}(sc(\tau_3) + sc(\tau_4)) + sc(\tau_3) - sc(\tau_4) \\ &\leq \frac{4\rho n}{1-\varepsilon}. \end{aligned}$$

Therefore,

$$\begin{aligned} g_A = N_A - b_A &\geq N_A - \frac{4\rho n}{1-\varepsilon} = (N_A - n(1-2\rho) + n(1-2\rho) + n(1-2\rho) - \frac{4\rho n}{1-\varepsilon} \\ &\geq b_A + b_B - \frac{4\rho n}{1-\varepsilon} + (N_A - n(1-2\rho)) + n(1-2\rho) - \frac{4\rho n}{1-\varepsilon} \\ &= b_A + b_B + (N_A - n(1-2\rho)) + n\left(1 - \frac{8\rho}{1-\varepsilon} - 2\rho\right) \\ &\geq b_A + b_B + (N_A - n(1-2\rho)) + n\left(1 - \frac{8\rho}{1-\varepsilon} - 2\rho\right) \\ &\geq b_A + b_B + (N_A - n(1-2\rho)) + 8\varepsilon n \geq b_A + b_B + (N_A - n(1-2\rho)) + T. \end{aligned}$$

The remainder of the proof follows exactly as is the proof of Theorem 4.4 from Braverman et al.'s work. □

Suppose that Alice uses an ε-edit-distance tree code $C : \Sigma_{in}^n \to \Sigma_{out}^n$ to send a message c to Bob. In the following theorem, we show that if $SD(c, \tilde{c}) < 1 - \varepsilon$, then Bob can decode Alice's message. Braverman et al. proved a weaker version of this theorem [BGMO17], but our revised definition of an edit-distance tree code easily admits the follow lemma.

Lemma 7.6.6 (Revision of Lemma 3.10 from [BGMO17]). *Let $C : \Sigma_{in}^n \to \Sigma_{out}^n$ be an ε-edit-distance tree code, and let $\tilde{c} \in \Sigma_{out}^m$. There exists at most one $c \in \cup_{i=1}^n C(\Sigma_{in}^n)[1,i]$ such that $SD(c,\tilde{c}) \leq 1-\varepsilon$.*

Proof. The proof follows exactly the same logic as the proof of Lemma 3.10 from [BGMO17]. The proof begins by assuming, for a contradiction, that there exist two messages $c, c' \in \cup_{i=1}^n C(\Sigma_{in}^n)[1,i]$ such that both $SD(c,\tilde{c}) \leq (1-\varepsilon)$ and $SD(c',\tilde{c}) \leq (1-\varepsilon)$. Then, the proof reaches its conclusion by the exact same logic. □

In Theorem 7.6.13, we will show that one can extend an ε-synchronization string S to a $(2\varepsilon - \varepsilon^2)$-edit-distance tree code. This implies the following corollary of Lemma 7.6.6.

Corollary 7.6.7. *Let $S \in \Sigma^n$ be an ε-synchronization string, and $c \in \Sigma^m$. There exists at most one prefix $\tilde{c}$ of S for which $SD(c,\tilde{c}) \leq (1-\varepsilon)^2$.*

7.6.2 Edit-distance tree codes and synchronization strings

We prove that under the revised definition, edit-distance tree codes still exist by relating edit-distance tree codes to Schulman's original tree codes. We introduce a method to obtain an edit-distance tree code using ordinary tree codes and synchronization strings. Intuitively, using synchronization strings and tree codes together, one can overcome the synchronization problem using the synchronization string and then overcome the rest of the decoding challenges using tree codes, which are standard tools for overcoming Hamming-type errors. Tree codes are defined as follows.

Definition 7.6.8 (Tree Codes, from [Sch96]). *A d-ary tree code over an alphabet Σ of distance parameter α and depth n is a d-ary tree of depth n in which every arc of the tree is labeled with a character from the alphabet Σ subject to the following condition. Let v_1 and v_2 be any two nodes at some common depth h in the tree. Let $h-l$ be the depth of their least common ancestor. Let $W(v_1)$ and $W(v_2)$ be the concatenation of the letters on the arcs leading from the root to v_1 and v_2 respectively. Then $\Delta(W(v_1), W(v_2)) \geq \alpha l$.*

Schulman [Sch93, Sch96] proved that for any $\alpha < 1$, there exists a d-ary tree code with distance α and infinite depth over an alphabet of size $(cd)^{1/(1-\alpha)}$, for some constant $c < 6$. We now prove that edit-distance tree codes can be obtained using synchronization strings and tree codes.

Theorem 7.6.9. *Let T be a tree code of depth n and distance parameter $(1-\alpha)$ and let S be an ε-synchronization string of length n. Let the tree T' be obtained by concatenating all edges on i^{th} level of T with $S[i]$. Then T' is a $(1-\varepsilon-\alpha)$-edit-distance tree code.*

Proof. To prove that T' is a $(1-\varepsilon-\alpha)$-edit-distance tree code, we show that T' does not contain any $(1-\varepsilon-\alpha)$-bad lambdas. Let A, B, D, and E be nodes in T' that form a lambda structure.

We refer to the string consisting of edge labels on the path from node A to D by AD and similarly we refer to the string consisting of labels on the path from B to E by BE. Let τ be the string matching characterizing the edit distance of AD and BE. We call i

a *matching position* if $\tau_1[i] = \tau_2[i]$. Further, we call i a *same-level matching position* if $\tau_1[i]$ and $\tau_2[i]$ correspond to symbols on the same level edges and *disparate-level matching position* otherwise.

By the definition of a tree code, the number of same-level matching positions is at most

$$\alpha \min\{|BD|, |BE|\} \leq \alpha(|AD| + |BE|).$$

Further, in Chapter 3 we showed that for any monotone matching between an ε-synchronization string and itself, i.e. a set of pairs $M = \{(a_1, b_1), \ldots, (a_m, b_m)\}$ where $a_1 < \cdots < a_m$, $b_1 < \cdots b_m$, and $S[a_i] = S[b_i]$, the number of pairs where $a_i \neq b_i$ is at most $\varepsilon|S|$. This means that the number of disparate-level matching positions is at most $\varepsilon(|AD| + |BE|)$. Therefore,

$$\begin{aligned} ED(AD, BE) &\geq |AD| + |BE| - \varepsilon(|AD| + |BE|) - \alpha(|AD| + |BE|) \\ &\geq (1 - \varepsilon - \alpha)(|AD| + |BE|). \end{aligned}$$

□

Note that Theorem 7.6.9 suggests a construction for edit-distance tree codes by simply constructing and concatenating an ordinary tree code and a synchronization string. As synchronization strings are efficiently constructible and tree codes can be constructed in sub-exponential time [Sch03, Bra12], this construction runs in sub-exponential time which improves over the construction of edit-distance tree codes from Braverman et al. [BGMO17].

The following theorems discuss further connections between synchronization strings and edit-distance tree codes. Theorems 7.6.11 and 7.6.13 show that edit-distance tree codes and synchronization strings are essentially similar combinatorial objects. In Theorem 7.6.11, we show that the edge labels along a monotonically down-going path in an edit-distance tree code form a synchronization string as though, in a manner, synchronization strings are one dimensional edit-distance tree codes. On the other hand, in Theorem 7.6.13, we show that for any synchronization string S, there is an edit-distance tree code that has S on one of its monotonically down-going paths.

Lemma 7.6.10. *In an ε-edit-distance tree code, for every three vertices X, Y, and Z where X is an ancestor of Y and Y is an ancestor of Z, we have that $ED(XY, YZ) \geq \left(1 - \varepsilon - \frac{1}{|XZ|}\right)|XZ|$.*

Proof. To show this, we set $A = X, B = Y, E = Z$ and denote the child of Y which is not in the path from Y to Z by D (see Figure 7.2). Then A, B, D, and E form a lambda in the tree (Definition 7.6.3). As ε-edit-distance tree codes are ε-bad lambda free, we know that $ED(AD, BE) \geq (1 - \varepsilon) \cdot (|AD| + |BE|)$.

Note that $ED(AD, BE) = ED(XD, YZ) \leq 1 + ED(XY, YZ)$ and $|AD| = |XY| + 1 > |XY|$, which means that $ED(XY, YZ) + 1 \geq (1 - \varepsilon)|XZ|$. Therefore, $ED(XY, YZ) \geq \left(1 - \varepsilon - \frac{1}{|XZ|}\right) \cdot |XZ|$. □

Using this property, one can obtain synchronization strings using monotonically down-going paths in a given edit-distance tree code as follows:

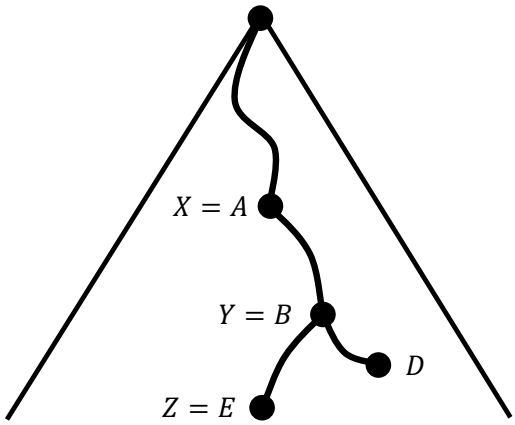


Figure 7.2: (A, B, D, E) form a Lambda structure.

Theorem 7.6.11. *Concatenating the symbols on any monotonically down-going path in an ε-edit-distance tree code with a string consisting of repetitions of $1, 2, \cdots, l$ gives an $\left(\varepsilon + \frac{1}{l}\right)$-synchronization string.*

Proof. Consider three indices $i < j < k$ in such string. If $k - i < l$, the $\left(\varepsilon + \frac{1}{l}\right)$-synchronization property holds as a result of the $1, \cdots, l, \cdots$ portion. Unless, the edit-distance path symbols satisfy $\left(\varepsilon + \frac{1}{l}\right)$-synchronization property as a result of Lemma 7.6.10. □

Theorem 7.6.11 results into the following corollary:

Corollary 7.6.12. *Existence of synchronization strings over constant-sized alphabets can be implied by Theorem 7.6.11 and the fact that edit-distance tree codes exist from [BGMO17]. However, the alphabet size would be exponentially large in terms of $\frac{1}{\varepsilon}$.*

Theorem 7.6.13. *Any ε-synchronization string in Σ_1^n can be extended to a $(2\varepsilon - \varepsilon^2)$-edit-distance tree code $C : \Sigma_{in} \to (\Sigma_{out} \cup \Sigma_1)^n$ using the ε-edit-distance tree code $C' : (\Sigma_{in}^n \to \Sigma_{out}^n)$ such that a monotonically down-going path on C is labeled as S.*

Proof. We simply replace the labels of edges on the rightmost path in the tree associated with C' to obtain C. We show that C is a valid $(2\varepsilon - \varepsilon^2)$-edit-distance tree code. To prove this, we need to verify that this tree does not contain an $(2\varepsilon - \varepsilon^2)$-bad lambda structure. In this proof, we set $\alpha = 1 - (2\varepsilon - \varepsilon^2) = (1 - \varepsilon)^2$ and $\alpha' = 1 - \varepsilon$.

Let A, B, D, E form a lambda structure in the corresponding tree of C. We will prove the claim under two different cases:

- BE **does not contain any edges from the rightmost path:** Assume that AD contains l edges from the rightmost path. In order to turn AD into BE, we must remove all l symbols associated with the rightmost path since BE has no symbols from Σ_1. Hence, $ED(AD, BE)$ is equal to l plus the edit distance of BE and AD after removal of those l edges. Note that removing l elements in AD and then converting

the remaining string $(\widetilde{AD})$ to BE is also a way of converting AD to BE in the original tree. Thus,

$$ED_{original}(AD, BE) \leq l + ED(\widetilde{AD}, BE) = ED_{modified}(AD + BE)$$

On the other hand, $ED_{original}(AD, BE) \geq \alpha' \cdot (|AD| + |BE|)$. Therefore,

$$ED_{modified}(AD, BE) \geq \alpha' \cdot (|AD| + |BE|).$$

- BE **contains some edges from the rightmost path:** It must be that all of AB and a non-empty prefix of BE, which we refer to as BX, both lie in the rightmost path in tree code (Fig. 7.3). Since the symbols in the rightmost path are from the alphabet Σ_1,

  $$ED(AD, BE) = ED(AB, BX) + ED(BD, XE).$$

  We consider the following two cases, where c is a constant we set later.

  1. $c \cdot (|AB| + |BX|) > |BD| + |XE|$:
     In this case,

     $$\begin{aligned} ED(AD, BE) &= ED(AB, BX) + ED(BD, XE) \\ &\geq ED(AB, BX) \\ &\geq (1 - \varepsilon) \cdot (|AB| + |BX|) \\ &\geq \frac{1 - \varepsilon}{c + 1} \cdot ((|AB| + |BX|) + (|BD| + |EX|)) \\ &\geq \frac{1 - \varepsilon}{c + 1} \cdot (|AD| + |BE|). \end{aligned}$$

  2. $c \cdot (|AB| + |BX|) \leq |BD| + |XE|$: Since (A, B, D, E) form a lambda structure,

     $$ED_{original}(AD, BE) \geq \alpha' \cdot (|AD| + |BE|)$$

     and therefore,

     $$\begin{aligned} ED_{modified}(AD, BE) &\geq ED_{modified}(BD, EX) + ED(AB, BX) \\ &\geq [ED_{original}(AD, BE) - (|AB| + |BX|)] + ED(AB, BX) \\ &\geq \alpha' \cdot (|AD| + |BE|) - (|AB| + |BX|) + (1 - \varepsilon) \cdot (|AB| + |BX|) \\ &= \alpha' \cdot (|AD| + |BE|) - \varepsilon(|AB| + |BX|) \\ &\geq \alpha' \cdot (|AD| + |BE|) - \frac{\varepsilon}{c + 1}(|AD| + |BE|) \\ &= \left(\alpha' - \frac{\varepsilon}{c + 1}\right)(|AD| + |BE|). \end{aligned}$$

Hence:

$$ED(AD, BE) \geq \min\left\{\alpha' - \frac{\varepsilon}{c + 1}, \frac{1 - \varepsilon}{c + 1}, \alpha'\right\}(|AD| + |BE|)$$

As $\alpha' = 1 - \varepsilon$, setting $c = \frac{\varepsilon}{1 - \varepsilon}$ gives that

$$ED(AD, BE) \geq (1 - \varepsilon)^2 \cdot (|AD| + |BE|)$$

which finishes the proof. □

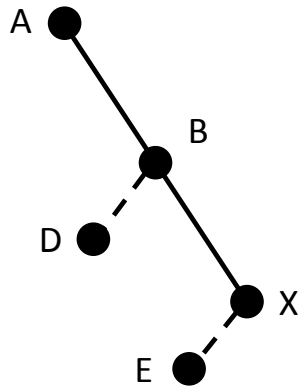


Figure 7.3: AD and BE both contain edges from the rightmost path. Straight lines represent the edges in the rightmost path and dashed ones represent other edges.

Chapter 8

Local Repositioning: Near-Linear Time

In this chapter, we present several algorithmic results for synchronization strings:

- We give a *deterministic, linear time synchronization string construction*, improving over an $O(n^5)$ time randomized construction in Chapter 4.
- We give a *deterministic construction of an infinite synchronization string* which outputs the first n symbols in $O(n)$ time.
- Both synchronization string constructions are *highly explicit*, i.e., the i^{th} symbol can be deterministically computed in $O(\log i)$ time.
- This chapter also introduces a generalized notion called *long-distance synchronization strings*. Such strings allow for *local and very fast* decoding. In particular only $O(\log^3 n)$ time and access to logarithmically many symbols is required to reposition any index.

This chapter also provides several applications for these improved synchronization strings:

- For any $\delta < 1$ and $\varepsilon > 0$, we provide an insertion-deletion code with rate $1 - \delta - \varepsilon$ which can correct any $\delta/3$ fraction of insertion and deletion errors in $O(n \log^3 n)$ time. This near linear computational efficiency is surprising given that we do not even know how to compute the (edit) distance between the decoding input and output in sub-quadratic time.
- We show that local decodability implies that error correcting codes constructed with long-distance synchronization strings can not only efficiently recover from δ fraction of insdel errors but, similar to [SZ99], also from any $O(\delta/\log n)$ fraction of *block transpositions and block replications*. These block corruptions allow arbitrarily long substrings to be swapped or replicated anywhere.
- We show that highly explicitness and local decoding allow for *infinite channel simulations with exponentially smaller memory and decoding time requirements*. These simulations can then be used to give the first *near-linear time interactive coding scheme for insertions and deletions*, similar to the result of [BN13] for Hamming errors.

8.1 Introduction: Our Results and Structure of this Chapter

This chapter provides drastically improved constructions of finite and infinite synchronization strings and a stronger synchronization-string-like property which allows for repositioning algorithms that are local and significantly faster. We, furthermore, give several applications for these results, including near-linear time insertion-deletion codes, a near-linear time coding scheme for interactive communication over insertion-deletion channels, exponentially better channel simulations in terms of time and memory, infinite channel simulations, and codes that can correct block transposition and block replication errors. We now give an overview of the main results and the overall structure of this chapter. We also put our result in the context of the related previous works.

8.1.1 Deterministic, Linear-Time, Highly Explicit Construction of Infinite Synchronization Strings

In Chapter 3, we introduced synchronization strings and gave an $O(n^5)$ time randomized construction for them. This construction could not be easily derandomized. In order to provide deterministic and explicit constructions of insertion deletion block codes, we introduced a strictly weaker notion called self-matching strings, showed that these strings could also be used for code constructions, and gave a deterministic $n^{O(1)}$ time construction for self-matching strings. Obtaining a deterministic construction for synchronization strings, however, was not addressed. Chapter 3 also showed the existence of infinite synchronization strings. This existence proof is highly non-constructive. In fact, even the existence of a computable infinite synchronization string was not shown; i.e., there was no algorithm that would compute the i^{th} symbol of some infinite synchronization string in finite time.

In this chapter, we give deterministic constructions of finite and infinite synchronization strings. Instead of going to a weaker notion, as done in Chapter 3, Section 8.3.1 introduces a stronger notion called long-distance synchronization strings. Interestingly, while the existence of these generalized synchronization strings can be shown with a similar Lovász local lemma based proof as for plain synchronization strings, this proof allows for an easier derandomization, which leads to a **deterministic polynomial time construction of (long-distance) synchronization strings**. Beyond this derandomization, the notion of long-distance synchronization strings turns out to be very useful and interesting in its own right, as will be shown later.

Next, two different boosting procedures, which make synchronization string constructions faster and more explicit, are given. The first boosting procedure, given in Section 8.3.3, leads to a **deterministic linear time synchronization string construction**. We remark that concurrently and independently Cheng, Li, and Wu obtained a deterministic $O(n \log^2 \log n)$ time synchronization string construction [CLW17].

Our second boosting step, which is introduced in Section 8.3.4, makes our synchronization string construction **highly-explicit**, i.e., allows to compute any position of a

synchronization string of length n in time $O(\log n)$. This highly-explicitness is a property of crucial importance in most of our new applications.

Lastly, in Section 8.3.5 we give a simple transformation which allows us to use any construction for finite length synchronization strings to give a construction of an **infinite synchronization string**. This transformation preserves highly-explicitness. Infinite synchronization strings are important for applications in which one has no prior bound on the running time of a system, such as, streaming codes, channel simulations, and interactive communications. Overall, we get the following simple theorem:

Theorem 8.1.1. *For any $0 < \varepsilon < 1$, there exists an infinite ε-synchronization string S over an alphabet of size $\varepsilon^{-O(1)}$ and a deterministic algorithm, which, for any i, takes $O(\log i)$ time to compute $S[i, i + \log i]$, i.e., the i^{th} symbol of S as well as the next $\log i$ symbols.*

Since any substring of an ε-synchronization string is also an ε-synchronization string itself, this infinite synchronization string construction also implies a deterministic linear time construction of finite synchronization strings which is fully parallelizable. In particular, for any positive integer n, there is a linear work parallel $\mathbf{NC}^1$ algorithm with depth $O(\log n)$ and $O(n/\log n)$ processors which computes the ε-synchronization string $S[1, n]$.

8.1.2 Long Distance Synchronization Strings and Fast Local Decoding

Section 8.4 shows that the long-distance property we introduced in Section 8.3.1, together with our highly explicit constructions from Section 8.3.3, lead to a much faster and highly local decoding (i.e., repositioning) procedure. In particular, to recover the position of an element in a stream that was indexed with a synchronization string, it suffices to look at only $O(\log n)$ previously received symbols. The repositioning takes only $O(\log^3 n)$ time and can be done in a streaming (i.e, online) fashion. This is significantly faster than the $O(n^3)$ streaming decoder or the $O(n^2)$ global repositioning algorithms given in Chapter 3.

This chapter, furthermore, gives several applications which demonstrate the power of these improved synchronization string constructions and the local decoding procedure.

8.1.3 Application: Codes Against InsDels, Block Transpositions and Replications

Near-Linear Time Decodable Codes

Fast encoding and decoding procedures for error correcting codes have been important and influencial in both theory and practice. For regular error correcting block codes, the celebrated expander code framework given by Sipser and Spielman [SS96] and in Spielman's thesis [Spi95] as well as later refinements by Alon, Edmonds, and Luby [AEL95] and Guruswami and Indyk [GI05, GI01] gave good ECCs with linear time encoding and decoding procedures. More recently, a work by Hemenway, Ron-Zewi, and Wooters [HRZW19]

achieved linear time decoding also for capacity achieving list-decodable and locally list-recoverable codes.

The synchronization string based insdel codes from Chapter 3 have linear encoding time complexity but quadratic decoding time complexity. As pointed out in Chapter 3, the latter seemed almost inherent to the harsher setting of insdel errors because *"in contrast to Hamming codes, even computing the distance between the received and the sent/decoded string is an edit distance computation. Edit distance computations, in general, do not run in sub-quadratic time, which is not surprising given the SETH-conditional lower bounds [BI18]"*. Surprisingly, our fast decoding procedure allows us to construct insdel codes with near linear decoding complexity:

Theorem 8.1.2. *For any $\delta < 1$ and $\varepsilon > 0$ there exists an **insdel error correcting block code** with rate $1 - \delta - \varepsilon$ that can correct from any $\delta/3$ fraction of insertions and deletions in $O(n \log^3 n)$ time. The encoding time is linear and the alphabet bit size is near-linear in $\frac{1}{\delta+\varepsilon}$.*

Note that, for any input string, the decoder finds the codeword that is closest to it in edit distance if a codeword with edit distance of at most $O(\delta n)$ exists. However, computing the distance between the input string and the codeword output by the decoder is an edit distance computation. Surprisingly, even now, we do not know of any sub-quadratic algorithm that can compute this distance between input and output of our decoder even though intuitively this seems to be much easier almost prerequisite step for the distance minimizing decoding problem itself. After all, decoding asks to find the closest (or a close) codeword to the input from an exponentially large set of codewords, which seems hard to do if one cannot even approximate the distance between the input and any particular codeword.

Application: High-Rate InsDel Codes that Efficiently Correct Block Transpositions and Replications

Section 8.5.2 gives another interesting application of our local decoding (repositioning) procedure. In particular, we show that local decodability directly implies that insdel ECCs constructed with our highly-explicit long-distance synchronization strings can not only efficiently recover from δ fraction of insdel errors but also from any $O(\delta/\log n)$ fraction of **block transpositions and block replications**. Block transpositions allow for arbitrarily long substrings to be swapped while a block replication allows for an arbitrarily long substring to be duplicated and inserted anywhere else. A similar result, albeit for block transpositions only, was shown by Schulman, Zuckerman [SZ99] for the efficient constant distance constant rate insdel codes given by them. They also show that the $O(\delta/\log n)$ resilience against block errors is optimal up to constant factors. A more recent work of Cheng et al. [CJLW19] also provides codes that can correct from insertions and deletions as well as block transpositions.

8.1.4 Application: Exponentially More Efficient Infinite Channel Simulations

In Chapter 7, we introduced the powerful notion of a channel simulation. In particular, we showed that for any adversarial one-way or two-way insdel channel one can put two *simulating agents* at the two ends of the channel, such that, to any two parties interacting with these agents, the agents simulate a much nicer Hamming channel which only introduces (a slightly larger fraction of) symbol erasures and symbol substitutions. To achieve this, these agents are required to know in advance for how many rounds R the channel would be used and require an amount of memory that is linear in terms of R. Furthermore, for each transmission at a time step t, the receiving agent would perform a $O(t^3)$ time computation. We show that using our locally-decodable highly-explicit long-distance synchronization strings can reduce both the memory requirement and the computation time complexity exponentially. In particular, each agent is only required to have $O(\log t)$ bits of memory (which is optimal because, at the very least, they need to store the length of the communication t) and their computations are done in $O(\log^3 t)$ time. Furthermore, due to our infinite synchronization string constructions, the channel simulations agents are not anymore required to know (an upper bound to) the length of the communication in advance. These drastic improvements make channel simulations significantly more useful and indeed potentially quite practical.

8.1.5 Application: Near-Linear Time Interactive Coding Schemes for InsDel Errors

Interactive coding schemes, as introduced by Schulman [Sch92, Sch96], allow to add redundancy to any interactive protocol between two parties in such a way that the resulting protocol becomes robust to noise in the communication. Interactive coding schemes that are robust to symbol substitutions have been intensely studied over the last few years [BR14, FGOS15, KR13, Hae14, BTK12, BN13, GH14, GMS14]. Similar to error correcting codes, the main parameters for an interactive coding scheme is the fraction of errors it can tolerate [Sch92, Sch96, BR14, FGOS15], its communication rate [KR13, Hae14], and its computational efficiency [BTK12, BN13, GH14, GMS14]. In particular, Brakerski and Kalai [BTK12] gave the first computationally efficient polynomial time interactive coding scheme. Brakerski and Naor [BN13] improved the time complexity to near-linear. Lastly, Ghaffari and Haeupler [GH14] gave a near-linear time interactive coding scheme that also achieved the optimal maximal robustness. More recently, interactive coding schemes that are robust to insertions and deletions have been introduced by Braverman, Gelles, Mao, and Ostrovsky [BGMO17]. Subsequently, Sherstov and Wu [SW19] gave a scheme with optimal error tolerance and we, as stated in Chapter 7, used channel simulations to give the first computationally efficient polynomial time interactive coding scheme for insdel errors. Our improved channel simulation can be used together with the coding scheme from [GH14] to directly get the first interactive coding scheme for insertions and deletions with a near-linear time complexity, i.e., the equivalent of the result of Brakerski and Naor [BN13] for insertions and deletions.

8.2 Definitions and Preliminaries

In this section, we review some of the notation and definitions from previous chapters that will be used throughout the rest of the chapter. We start with recapitulating the definition of the relative suffix distance and its metric property from Chapter 3.

Definition 8.2.1 (Relative Suffix Distance (Definition 3.4.2))**.** *For any two strings $S, S' \in \Sigma^*$ we define their relative suffix distance* RSD *as follows:*

$$\mathrm{RSD}(S, S') = \max_{k>0} \frac{\mathrm{ED}\left(S(|S|-k, |S|], S'(|S'|-k, |S'|]\right)}{2k}$$

Lemma 8.2.2 (Lemma 3.4.3)**.** *For any strings S_1, S_2, S_3 we have*

- ***Symmetry:*** $\mathrm{RSD}(S_1, S_2) = \mathrm{RSD}(S_2, S_1)$,
- ***Non-Negativity and Normalization:*** $0 \leq \mathrm{RSD}(S_1, S_2) \leq 1$,
- ***Identity of Indiscernibles:*** $\mathrm{RSD}(S_1, S_2) = 0 \Leftrightarrow S_1 = S_2$, *and*
- ***Triangle Inequality:*** $\mathrm{RSD}(S_1, S_3) \leq \mathrm{RSD}(S_1, S_2) + \mathrm{RSD}(S_2, S_3)$.

In particular, RSD *defines a metric on any set of strings.*

We now briefly review some fundamental concepts regarding synchronization strings and the indexing technique presented in previous chapters. In short, synchronization strings allow communicating parties to protect against synchronization errors by indexing their messages without blowing up the communication rate. The general idea of coding schemes introduced and utilized in previous chapters was to *index* communicated symbols in the sender side with symbols of a synchronization string and then *guess* the actual position of received symbols on the other end using the attached indices – a procedure that we named *repositioning*.

Generally speaking, for a communication of length n that suffers from δ-fraction of synchronization errors, such a pair of index string and repositioning algorithm is called an (n, δ)*-indexing solution*. In this setup, a symbol which is sent by Alice and is received by Bob without being deleted by the adversary is called a *successfully transmitted* symbol. Ideally, a indexing solution is supposed to correctly figure out the position of as many successfully transmitted symbols as possible. The measure of *misdecodings* was introduced in Chapter 3 to evaluate the quality of a (n, δ)-indexing solution as the number of successfully transmitted symbols that an algorithm might not decoded correctly. An indexing algorithm is called to be *streaming* if its output for a particular received symbol depends only on the symbols that have been received before it. We will make use of the following theorem from Chapter 3 that summarizes how indexing solutions and ECCs can be used to construct insdel codes.

Theorem 8.2.3 (Theorem 3.3.2 from Chapter 3)**.** *Given a synchronization string S over alphabet Σ_S with an (efficient) decoding algorithm $\mathcal{D}_S$ guaranteeing at most k misdecodings and decoding complexity $T_{\mathcal{D}_S}(n)$ and an (efficient) ECC $\mathcal{C}$ over alphabet $\Sigma_{\mathcal{C}}$ with rate $R_{\mathcal{C}}$,*

encoding complexity $T_{\mathcal{E}_\mathcal{C}}$, and decoding complexity $T_{\mathcal{D}_\mathcal{C}}$ that corrects up to $n\delta + 2k$ half-errors, one obtains an insdel code that can be (efficiently) decoded from up to $n\delta$ insertions and deletions. The rate of this code is at least

$$\frac{R_\mathcal{C}}{1 + \frac{\log|\Sigma_S|}{\log|\Sigma_\mathcal{C}|}}$$

The encoding complexity remains $T_{\mathcal{E}_\mathcal{C}}$, the decoding complexity is $T_{\mathcal{D}_\mathcal{C}} + T_{\mathcal{D}_S}(n)$ and the preprocessing complexity of constructing the code is the complexity of constructing $\mathcal{C}$ and S.

In Chapter 3, we introduced synchronization strings, a family of strings parametrized by ε that were shown to be good candidates for forming indexing solutions. Synchronization strings exist over alphabets of constant size in terms of communication length n and dependent merely on parameter ε. We review the formal definition of synchronization strings here.

Definition 8.2.4 (ε-Synchronization String (Definition 3.4.4))**.** *String $S \in \Sigma^n$ is an ε-synchronization string if for every $1 \leq i < j < k \leq n+1$ we have that $ED(S[i,j), S[j,k)) > (1-\varepsilon)(k-i)$. We call the set of prefixes of such a string an ε-synchronization code.*

We will make use of the following repositioning algorithm for synchronization strings from Chapter 3.

Theorem 8.2.5 (Simplified version of Theorem 3.5.14)**.** *There is a repositioning algorithm (Algorithm 3) for an ε-synchronization string of length n which guarantees decoding with up to $O(n\sqrt{\varepsilon})$ misdecodings and runs in $O(n^2/\sqrt{\varepsilon})$ time.*

8.3 Highly Explicit Constructions of Long-Distance and Infinite ε-Synchronization Strings

We start this section by introducing a generalized notion of synchronization strings in Section 8.3.1 and then provide a deterministic efficient construction for them in Section 8.3.2. In Section 8.3.3, we provide a boosting step which speeds up the construction to linear time in Theorem 8.3.7. In Section 8.3.4, we use the linear time construction to obtain a linear-time high-distance insdel code (Theorem 8.3.14) and then use another boosting step to obtain a highly-explicit linear-time construction for long-distance synchronization strings in Theorem 8.3.15. We provide similar construction for infinite synchronization strings in Section 8.3.5. A pictorial representation of the flow of theorems and lemmas in this section can be found in Figure 8.1.

8.3.1 Long-Distance Synchronization Strings

The existence of synchronization strings is proven in Chapter 3 using an argument based on Lovász local lemma. This lead to an efficient randomized construction for synchronization strings which cannot be easily derandomized. Instead, the authors introduced

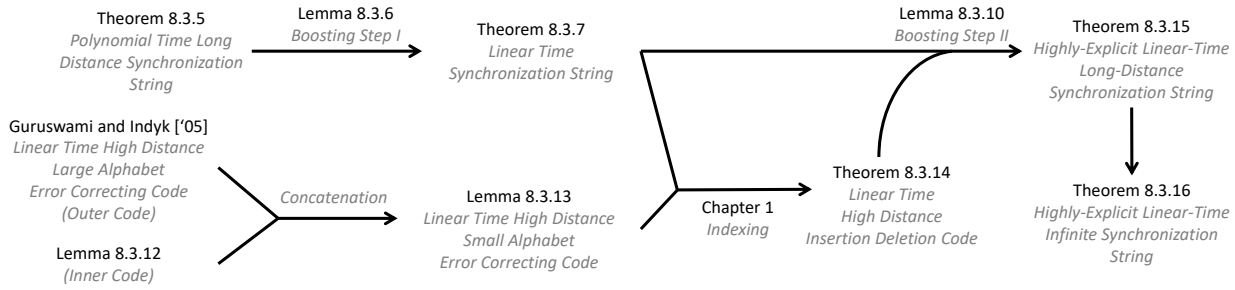


Figure 8.1: Schematic flow of Theorems and Lemmas of Section 8.3

the weaker notion of self-matching strings and gave a deterministic construction for them. Interestingly, in this chapter, we introduce a revised notion, denoted by $f(l)$-*distance* ε-*synchronization strings*, which generalizes ε-synchronization strings and allows for a deterministic construction.

Note that the synchronization string property poses a requirement on the edit distance of neighboring substrings. $f(l)$-distance ε-synchronization string property extends this requirement to any pair of intervals that are close by. More formally, any two intervals of aggregated length l that are of distance $f(l)$ or less have to satisfy the edit distance property in this generalized notion.

Definition 8.3.1 ($f(l)$-distance ε-synchronization string)**.** *String $S \in \Sigma^n$ is an $f(l)$-distance ε-synchronization string if for every $1 \leq i < j \leq i' < j' \leq n+1$ we have that $ED\left(S[i,j), S[i',j')\right) > (1-\varepsilon)l$ if $i' - j \leq f(l)$ where $l = j + j' - i - i'$.*

It is noteworthy to mention that the constant function $f(l) = 0$ gives the original ε-synchronization strings. Chapter 3 studied the existence and construction of synchronization strings for this case. In particular, they have shown that arbitrarily long ε-synchronization strings exist over an alphabet that is polynomially large in terms of ε^{-1}. Besides $f(l) = 0$, there are several other functions that might be of interest in this context.

One can show that, as we do in Section 8.8, for any polynomial function $f(l)$, arbitrarily long $f(l)$-distance ε-synchronization strings exist over alphabet sizes that are polynomially large in terms of ε^{-1}. Also, for exponential functions, these strings exist over exponentially large alphabets in terms of ε^{-1} but not over sub-exponential alphabet sizes. Finally, if function f is super-exponential, $f(l)$-distance ε-synchronization strings do not exist over any constant size alphabet. The similar question of constructing infinite binary strings that avoid identical substrings of length n with exponential distance in terms of n have been studied by Beck [Bec84].

While studying existence, construction, and alphabet sizes of $f(l)$-distance ε-synchronization strings might be of interest by its own, we will show that having synchronization string edit distance guarantee for pairs of intervals that are exponentially far in terms of their aggregated length is of significant interest as it leads to improvements over applications of ordinary synchronization strings described in Chapters 3 and 7 from several aspects. Even though distance function $f(l) = c^l$ provides such property, throughout the

rest of this paper, we will focus on a variant of it, i.e., $f(l) = n \cdot \mathbb{1}_{l>c\log n}$ which allows polynomial-sized alphabet. $\mathbb{1}_{l>c\log n}$ is the indicator function for $l > c\log n$, i.e., is one if $l > c\log n$ and zero otherwise

To compare distance functions $f(l) = c^l$ and $f(l) = n \cdot \mathbb{1}_{l>c\log n}$, note that the first one allows intervals to be exponentially far away in their total length. In particular, intervals of length $l > c\log n$ or larger can be arbitrarily far away. The second function only asks for the guarantee over large intervals and does not strengthen the ε-synchronization property for smaller intervals. We refer to the latter as *c-long-distance ε-synchronization string* property.

Definition 8.3.2 (c-long-distance ε-synchronization strings)**.** *We call $n \cdot \mathbb{1}_{l>c\log n}$-distance ε-synchronization strings* c-long-distance ε-synchronization strings.

8.3.2 Efficient Construction of Long-Distance Synchronization Strings

An LLL-based proof for the existence of ordinary synchronization strings has been provided in Chapter 3. Here we provide a similar technique along with the deterministic algorithm for Lovász local lemma from Chandrasekaran et al. [CGH13] to prove the existence and give a deterministic polynomial-time construction of strings that satisfy this quality over an alphabet of size $\varepsilon^{-O(1)}$.

Before giving this proof right away, we first show a property of these strings which allows us to simplify the proof and, more importantly, get a deterministic algorithm using deterministic algorithms for Lovász local lemma from Chandrasekaran et al. [CGH13].

Lemma 8.3.3. *If S is a string and there are two intervals $i_1 < j_1 \leq i_2 < j_2$ of total length $l = j_1 - i_1 + j_2 - i_2$ and $\mathrm{ED}(S[i_1, j_1), S[i_2, j_2)) \leq (1-\varepsilon)l$ then there also exists intervals $i_1 \leq i'_1 < j'_1 \leq i'_2 < j'_2 \leq j_2$ of total length $l' \in \{\lceil l/2\rceil - 1, \lceil l/2\rceil, \lceil l/2\rceil + 1\}$ with $\mathrm{ED}(S[i'_1, j'_1), S[i'_2, j'_2)) \leq (1-\varepsilon)l'$.*

Proof. As $\mathrm{ED}(S[i_1, j_1), [i_2, j_2)) \leq (1-\varepsilon)l$, there has to be a common subsequence of length $m \geq \frac{\varepsilon l}{2}$ between $S[i_1, j_1)$ and $S[i_2, j_2)$ locating at indices $a_1 < a_2 < \cdots < a_m$ and $b_1 < b_2 < \cdots < b_m$ respectively. We call $M = \{(a_1, b_1), \cdots, (a_m, b_m)\}$ a monotone matching from $S[i_1, j_1)$ to $S[i_2, j_2)$. Let $1 \leq i \leq m$ be the largest number such that $|S[i_1, a_i]| + |S[i_2, b_i]| \leq \lceil l/2\rceil$. It is easy to verify that there are integers $a_i < k_1 \leq a_{i+1}$ and $b_i < k_2 \leq b_{i+1}$ such that $|S[i_1, k_1)| + |S[i_2, k_2)| \in \{\lceil l/2\rceil - 1, \lceil l/2\rceil\}$.

Therefore, we can split the pair of intervals $(S[i_1, j_1), S[i_2, j_2))$ into two pairs of intervals $(S[i_1, k_1), S[i_2, k_2))$ and $(S[k_1, j_1), S[k_2, j_2))$ such that each pair of the matching M falls into exactly one of these pairs. Hence, in at least one of those pairs, the size of the matching is larger than $\frac{\varepsilon}{2}$ times the total length. This gives that the edit distance of those pairs is less than $1 - \varepsilon$ and finishes the proof. □

Lemma 8.3.3 shows that if there is a pair of intervals of total length l that have small relative edit distance, we can find a pair of intervals of size $\{\lceil l/2\rceil - 1, \lceil l/2\rceil, \lceil l/2\rceil + 1\}$ which have small relative edit distance as well. Now, let us consider a string S with a pair of

intervals that violate the c-long distance ε-synchronization property. If the total length of the intervals exceed $2c\log n$, using Lemma 8.3.3 we can find another pair of intervals of almost half the total length which still violate the c-long distance ε-synchronization property. Note that as their total length is longer than $c\log n$, we do not worry about the distance of those intervals. Repeating this procedure, we can eventually find a pair of intervals of a total length between $c\log n$ and $2c\log n$ that violate the c-long distance ε-synchronization property. More formally, we can derive the following statement by Lemma 8.3.3.

Corollary 8.3.4. *If S is a string which satisfies the c-long-distance ε-synchronization property for any two non-adjacent intervals of total length $2c\log n$ or less, then it satisfies the property for all pairs of non-adjacent intervals.*

Proof. Suppose, for the sake of contradiction, that there exist two intervals of total length $2\log_c n$ or more that violate the c-long-distance ε-synchronization property. Let $[i_1, j_1)$ and $[i_2, j_2)$ where $i_1 < j_1 \leq i_2 < j_2$ be two intervals of the smallest total length $l = j_1 - i_1 + j_2 - i_2$ larger than $2\log_c n$ (breaking ties arbitrarely) for which $\mathrm{ED}(S[i_1, j_1), [i_2, j_2)) \leq (1-\varepsilon)l$. By Lemma 8.3.3 there exists two intervals $[i'_1, j'_1)$ and $[i'_2, j'_2)$ where $i'_1 < j'_1 \leq i'_2 < j'_2$ of total length $l' \in [l/2, l)$ with $\mathrm{ED}(S[i'_1, j'_1), [i'_2, j'_2)) \leq (1-\varepsilon)l$. If $l' \leq 2\log_c n$, the assumption of c-long-distance ε-synchronization property holding for intervals of length $2\log_c n$ or less is contradicted. Unless, $l' > 2\log_c n$ that contradicts the minimality of our choice of l. □

Theorem 8.3.5. *For any $0 < \varepsilon < 1$ and every n there is a deterministic $n^{O(1)}$ time algorithm for computing a $c = O(1/\varepsilon)$-long-distance ε-synchronization string over an alphabet of size $O(\varepsilon^{-4})$.*

Proof. To prove this, we will make use of the Lovász local lemma and deterministic algorithms proposed for it in [CGH13]. We generate a random string R over an alphabet of size $|\Sigma| = O(\varepsilon^{-2})$ and define bad event B_{i_1,l_1,i_2,l_2} as the event of intervals $[i_1, i_1 + l_1)$ and $[i_2, i_2 + l_2)$ violating the $O(1/\varepsilon)$-long-distance synchronization string property over intervals of total length $2/\varepsilon^2$ or more. In other words, B_{i_1,l_1,i_2,l_2} occurs if and only if $\mathrm{ED}(R[i_1, i_1+l_1), R[i_2, i_2+l_2)) \leq (1-\varepsilon)(l_1+l_2)$. Note that by the definition of c-long-distance ε-synchronization strings, B_{i_1,l_1,i_2,l_2} is defined for (i_1, l_1, i_2, l_2)s where either $l_1 + l_2 \geq c\log n$ and $i_1 + l_1 \leq i_2$ or $2/\varepsilon^2 < l_1 + l_2 < c\log n$ and $i_2 = i_1 + l_1$. We aim to show that for large enough n, with non-zero probability, none of these bad events happen. This will prove the existence of a string that satisfies $c = O(1/\varepsilon)$-long-distance ε-synchronization strings for all pairs of intervals that are of total length $2/\varepsilon^2$ or more. To turn this string into a $c = O(1/\varepsilon)$-long-distance ε-synchronization strings, we simply concatenate it, symbol-by-symbol, with a string consisting of repetitions of $1, \cdots, 2\varepsilon^{-2}$, i.e., $1, 2, \cdots, 2\varepsilon^{-2}, 1, 2, \cdots, 2\varepsilon^{-2}, \cdots$. This will take care of the edit distance requirement for neighboring intervals with total length smaller than $2\varepsilon^{-2}$.

Note that using Lemma 8.3.3 and by a similar argument as in Claim 8.3.4, we only need to consider bad events where $l_1 + l_2 \leq 2c\log n$. As the first step, note that B_{i_1,l_1,i_2,l_2} happens only if there is a common subsequence of length $\varepsilon(l_1 + l_2)/2$ or more between $R[i_1, i_1 + l_1)$ and $R[i_2, i_2 + l_2)$. Hence, the union bound gives that

$$\begin{aligned}
\Pr\{B_{i_1,l_1,i_2,l_2}\} &\leq \binom{l_1}{\varepsilon(l_1+l_2)/2}\binom{l_1}{\varepsilon(l_1+l_2)/2}|\Sigma|^{-\frac{\varepsilon(l_1+l_2)}{2}} \\
&\leq \left(\frac{l_1 e}{\varepsilon(l_1+l_2)/2}\right)^{\varepsilon(l_1+l_2)/2}\left(\frac{l_2 e}{\varepsilon(l_1+l_2)/2}\right)^{\varepsilon(l_1+l_2)/2}|\Sigma|^{-\frac{\varepsilon(l_1+l_2)}{2}} \\
&= \left(\frac{2e\sqrt{l_1 l_2}}{\varepsilon(l_1+l_2)\sqrt{|\Sigma|}}\right)^{\varepsilon(l_1+l_2)} \\
&\leq \left(\frac{el}{\varepsilon l\sqrt{|\Sigma|}}\right)^{\varepsilon l} = \left(\frac{e}{\varepsilon\sqrt{|\Sigma|}}\right)^{\varepsilon l}
\end{aligned}$$

where $l = l_1 + l_2$. In order to apply LLL, we need to find real numbers $x_{i_1,l_1,i_2,l_2} \in [0,1]$ such that for any B_{i_1,l_1,i_2,l_2}

$$\Pr\{B_{i_1,l_1,i_2,l_2}\} \leq x_{i_1,l_1,i_2,l_2} \prod_{[S[i_1,i_1+l_1)\cup S[i_2,i_2+l_2)]\cap[S[i'_1,i'_1+l'_1)\cup S[i'_2,i'_2+l'_2)]\neq\emptyset} (1 - x_{i'_1,l'_1,i'_2,l'_2}) \tag{8.1}$$

We eventually want to show that our LLL argument satisfies the conditions required for polynomial-time deterministic algorithmic LLL specified in [CGH13]. Namely, it suffices to certify two other properties in addition to (8.1). The first additional requirement is to have each bad event in LLL depend on up to logarithmically many variables and the second is to have (8.1) hold with a constant exponential slack. The former is clearly true as our bad events consist of pairs of intervals each of which is of a length between $c\log n$ and $2c\log n$. To have the second requirement, instead of (8.1) we find $x_{i_1,l_1,i_2,l_2} \in [0,1]$ that satisfy the following stronger property.

$$\Pr\{B_{i_1,l_1,i_2,l_2}\} \leq \left[x_{i_1,l_1,i_2,l_2} \prod_{[S[i_1,i_1+l_1)\cup S[i_2,i_2+l_2)]\cap[S[i'_1,i'_1+l'_1)\cup S[i'_2,i'_2+l'_2)]\neq\emptyset} (1 - x_{i'_1,l'_1,i'_2,l'_2})\right]^{1.01} \tag{8.2}$$

Any small constant can be used as slack. We pick 1.01 for the sake of simplicity. We propose $x_{i_1,l_1,i_2,l_2} = D^{-\varepsilon(l_1+l_2)}$ for some $D > 1$ to be determined later. D has to be chosen such that for any i_1, l_1, i_2, l_2 and $l = l_1 + l_2$:

$$\left(\frac{e}{\varepsilon\sqrt{|\Sigma|}}\right)^{\varepsilon l} \leq \left[D^{-\varepsilon l} \prod_{[S[i_1,i_1+l_1)\cup S[i_2,i_2+l_2)]\cap[S[i'_1,i'_1+l'_1)\cup S[i'_2,i'_2+l'_2)]\neq\emptyset} \left(1 - D^{-\varepsilon(l'_1+l'_2)}\right)\right]^{1.01} \tag{8.3}$$

Note that:

$$D^{-\varepsilon l} \prod_{[S[i_1,i_1+l_1)\cup S[i_2,i_2+l_2)]\cap[S[i'_1,i'_1+l'_1)\cup S[i'_2,i'_2+l'_2)]\neq\emptyset} \left(1 - D^{-\varepsilon(l'_1+l'_2)}\right)$$

$$
\begin{aligned}
&\geq D^{-\varepsilon l}\prod_{l'=c\log n}^{2c\log n}\prod_{l_1'=1}^{l'}\left(1-D^{-\varepsilon l'}\right)^{\left[(l_1+l_1')+(l_1+l_2')+(l_2+l_1')+(l_2+l_2')\right]n}\\
&\quad\times\prod_{l''=1/\varepsilon^2}^{c\log n}\left(1-D^{-\varepsilon l''}\right)^{l+l''} \qquad (8.4)\\
&= D^{-\varepsilon l}\prod_{l'=c\log n}^{2c\log n}\prod_{l_1'=1}^{l'}\left(1-D^{-\varepsilon l'}\right)^{4(l+l')n}\times\prod_{l''=1/\varepsilon^2}^{c\log n}\left(1-D^{-\varepsilon l''}\right)^{l+l''}\\
&= D^{-\varepsilon l}\prod_{l'=c\log n}^{2c\log n}\left(1-D^{-\varepsilon l'}\right)^{4l'(l+l')n}\times\left[\prod_{l''=1/\varepsilon^2}^{c\log n}\left(1-D^{-\varepsilon l''}\right)\right]^{l}\times\prod_{l''=1/\varepsilon^2}^{c\log n}\left(1-D^{-\varepsilon l''}\right)^{l''}\\
&\geq D^{-\varepsilon l}\left(1-\sum_{l'=c\log n}^{2c\log n}(4l'(l+l')n)\,D^{-\varepsilon l'}\right)\\
&\quad\times\left[1-\sum_{l''=1/\varepsilon^2}^{c\log n}D^{-\varepsilon l''}\right]^{l}\times\left(1-\sum_{l''=1/\varepsilon^2}^{c\log n}l''D^{-\varepsilon l''}\right) \qquad (8.5)\\
&\geq D^{-\varepsilon l}\left(1-\sum_{l'=c\log n}^{2c\log n}(4\cdot 2c\log n(2c\log n+2c\log n)n)\,D^{-\varepsilon l'}\right)\\
&\quad\times\left[1-\sum_{l''=1/\varepsilon^2}^{\infty}D^{-\varepsilon l''}\right]^{l}\times\left(1-\sum_{l''=1/\varepsilon^2}^{\infty}l''D^{-\varepsilon l''}\right)\\
&= D^{-\varepsilon l}\left(1-\sum_{l'=c\log n}^{2c\log n}\left(32c^2n\log^2 n\right)D^{-\varepsilon l'}\right)\times\left[1-\frac{D^{-\varepsilon\cdot 1/\varepsilon^2}}{1-D^{-\varepsilon}}\right]^{l}\\
&\quad\times\left(1-\frac{D^{-\varepsilon\cdot 1/\varepsilon^2}(D^{-\varepsilon}+1/\varepsilon^2-D^{-\varepsilon}/\varepsilon^2)}{(1-D^{-\varepsilon})^2}\right)\\
&\geq D^{-\varepsilon l}\left(1-32c^3n\log^3 nD^{-\varepsilon c\log n}\right)\left[1-\frac{D^{-1/\varepsilon}}{1-D^{-\varepsilon}}\right]^{l}\\
&\quad\times\left(1-\frac{D^{-1/\varepsilon}(D^{-\varepsilon}+1/\varepsilon^2-D^{-\varepsilon}/\varepsilon^2)}{(1-D^{-\varepsilon})^2}\right) \qquad (8.6)
\end{aligned}
$$

To justify equation (8.4), note that there are two kinds of bad events that might intersect B_{i_1,l_1,i_2,l_2}. The first product term is considering all pairs of long intervals of length l_1' and l_2' where $l_1+l_2\geq c\log n$ that overlap a fixed pair of intervals of length l_1 and l_2. The number of such intervals is at most $[(l_1+l_1')+(l_1+l_2')+(l_2+l_1')+(l_2+l_2')]\,n$. The second one is considering short neighboring pairs of intervals ($\varepsilon^{-2}\leq l''=l_1''+l_2''\leq c\log n$).

Equation (8.5) is a result of the following inequality for $0<x,y<1$:

$$(1-x)(1-y)>1-x-y.$$

We choose $D = 2$ and $c = 2/\varepsilon$. Note that $\lim_{\varepsilon\to 0} \frac{2^{-1/\varepsilon}(2^{-\varepsilon}+1/\varepsilon^2-2^{-\varepsilon}/\varepsilon^2)}{(1-2^{-\varepsilon})^2} = 0$. So, for small enough ε, $\frac{2^{-1/\varepsilon}}{1-2^{-\varepsilon}} < \frac{1}{2}$. Also, for $D = 2$ and $c = 2/\varepsilon$,

$$32c^3 n \log^3 n D^{-\varepsilon c \log n} = \frac{2^8}{\varepsilon^3} \cdot \frac{\log^3 n}{n} = o(1).$$

Finally, one can verify that for small enough ε, $1 - \frac{2^{-1/\varepsilon}}{1-2^{-\varepsilon}} > 2^{-\varepsilon}$. Therefore, for sufficiently small ε and sufficiently large n, (8.6) is satisfied if the following is satisfied.

$$D^{-\varepsilon l} \prod_{[S[i_1,i_1+l_1)\cup S[i_2,i_2+l_2)]\cap[S[i'_1,i'_1+l'_1)\cup S[i'_2,i'_2+l'_2)]\neq\emptyset} \left(1 - D^{-\varepsilon(l'_1+l'_2)}\right) \tag{8.7}$$

$$\geq \quad 2^{-\varepsilon l}\left(1-\frac{1}{2}\right)\left(2^{-\varepsilon}\right)^l\left(1-\frac{1}{2}\right) \geq \frac{4^{-\varepsilon l}}{4} \tag{8.8}$$

So, for LLL to work, the following have to be satisfied.

$$\left(\frac{e}{\varepsilon\sqrt{|\Sigma|}}\right)^{\frac{\varepsilon l}{1.01}} \leq \frac{4^{-\varepsilon l}}{4} \Leftrightarrow 4 \leq \left(\frac{\varepsilon\sqrt{|\Sigma|}}{e4^{1.01}}\right)^{\frac{\varepsilon l}{1.01}}$$

$$\Leftarrow \quad 4 \leq \left(\frac{\varepsilon\sqrt{|\Sigma|}}{e4^{1.01}}\right)^{\frac{\varepsilon\cdot 1/\varepsilon^2}{1.01}} \Leftrightarrow \frac{4^{2.02(1+\varepsilon)}e^2}{\varepsilon^2} \leq |\Sigma|$$

Therefore, for $|\Sigma| = \frac{4^{4.04}e^2}{\varepsilon^2} = O(\varepsilon^{-2})$, the deterministic LLL conditions hold. This finishes the proof. □

8.3.3 Boosting I: Linear Time Construction of Synchronization Strings

Next, we provide a simple boosting step which allows us to polynomially speed up any ε-synchronization string construction. Essentially, we propose a way to construct an $O(\varepsilon)$-synchronization string of length $O_\varepsilon(n^2)$ having an ε-synchronization string of length n.

Lemma 8.3.6. *Fix an even $n \in \mathbb{N}$ and $\gamma > 0$ such that $\gamma n \in \mathbb{N}$. Suppose $S \in \Sigma^n$ is an ε-synchronization string. The string $S' \in \Sigma'^{\gamma n^2}$ with $\Sigma' = \Sigma^3$ and*

$$S'[i] = \left(S[i \bmod n], S[(i + n/2) \bmod n], S\left[\left\lfloor\left\lfloor \frac{i}{\gamma n}\right\rfloor\right\rfloor\right]\right)$$

is an $(\varepsilon + 6\gamma)$-synchronization string of length γn^2.

Proof. Intervals of length at most $n/2$ lie completely within a copy of S and thus have the ε-synchronization property. For intervals of size l larger than $n/2$ we look at the synchronization string which is blown up by repeating each symbol γn times (i.e., third element of the concatenation). Ensuring that both sub-intervals contain complete blocks

changes the edit distance by at most $3\gamma n$ and thus by at most $6\gamma l$. Once only complete blocks are contained we use the observation that the longest common subsequence of any two strings becomes exactly a factor k larger if each symbols is repeated k times in each string. This means that the relative edit distance does not change and is thus at least ε. Overall this results in the $(\varepsilon + 6\gamma)$-synchronization string property to hold for large intervals in S'. □

We use this step to speed up the polynomial time deterministic ε-synchronization string construction in Theorem 8.3.5 to linear time.

Theorem 8.3.7. *There exists an algorithm that, for any $0 < \varepsilon < 1$, constructs an ε-synchronization string of length n over an alphabet of size $\varepsilon^{-O(1)}$ in $O(n)$ time.*

Proof. Note that if one takes an ε'-synchronization strings of length n' and applies the boosting step in Theorem 8.3.6 k times with parameter γ, he would obtain a $(\varepsilon' + 6k\gamma)$-synchronization string of length $\gamma^{2^k-1}n^{2^k}$.

For any $0 < \varepsilon < 1$, Theorem 8.3.5 gives a deterministic algorithm for constructing an ε-synchronization string over an alphabet $O(\varepsilon^{-4})$ that takes $O(n^T)$ time for some constant T independent of ε and n. We use the algorithm in Theorem 8.3.5 to construct an $\varepsilon' = \frac{\varepsilon}{2}$ synchronization string of length $n' = \frac{n^{1/T}}{\gamma}$ for $\gamma = \frac{\varepsilon}{12 \log T}$ over an alphabet of size $O(\varepsilon^{-4})$ in $O(n'^T) = O(n)$ time. Then, we apply boosting step I $k = \log T$ times with $\gamma = \frac{\varepsilon}{12 \log T}$ to get an $(\varepsilon' + 6\gamma \log T = \varepsilon)$-synchronization string of length $\gamma^{T-1}n'^T \geq n$. As boosting step have been employed constant times, the eventual alphabet size will be $\varepsilon^{-O(1)}$ and the run time is $O(n)$. □

8.3.4 Boosting II: Explicit Constructions for Long-Distance Synchronization Strings

We start this section by a discussion of *explicitness* quality of synchronization string constructions. In addition to the time complexity of synchronization strings' constructions, an important quality of a construction that we take into consideration for applications that we will discuss later is explicitness or, in other words, how fast one can calculate a particular symbol of a synchronization string.

Definition 8.3.8 ($T(n)$-explicit construction)**.** *If a synchronization string construction algorithm can compute ith index of the string it is supposed to find, i.e., $S[i]$, in $T(n)$ we call it an $T(n)$-*explicit *algorithm.*

We are particularly interested in cases where $T(n)$ is polylogarithmically large in terms of n. For such $T(n)$, a $T(n)$-explicit construction implies a near-linear construction of the entire string as one can simply compute the string by finding out symbols one by one in $n \cdot T(n)$ overall time. We use the term *highly-explicit* to refer to $O(\log n)$-explicit constructions.

We now introduce a boosting step in Lemma 8.3.10 that will lead to explicit constructions of (long-distance) synchronization strings. Lemma 8.3.10 shows that, using a

high-distance insertion-deletion code, one can construct strings that satisfy the requirement of long-distance synchronization strings for every pair of substrings that are of total length $\Omega_\varepsilon(\log n)$ or more. Having such a string, one can construct a $O_\varepsilon(1)$-long-distance ε-synchronization string by simply concatenating the outcome of Lemma 8.3.10 with repetitions of an $O_\varepsilon(\log n)$-long ε-synchronization string.

This boosting step is deeply connected to our new definition of long-distance ε-synchronization strings. In particular, we observe the following interesting connection between insertion-deletion codes and long-distance ε-synchronization strings.

Lemma 8.3.9. *If S is a c-long-distance ε-synchronization string over an alphabet of size q where $c = \Theta(1)$ then $\mathcal{C} = \{S(i \cdot c \log n, (i+1) \cdot c \log n] | 0 \leq i < \frac{n}{c \log n} - 1\}$ is an insdel error correcting code with minimum distance at least $1 - \varepsilon$ and constant rate $\Omega_q(1)$. Further, if any substring $S[i, i + \log n]$ is computable in $O(\log n)$ time, $\mathcal{C}$ has a linear encoding time.*

Proof. The distance follows from the definition of long-distance ε-synchronization strings. The rate follows because the rate R is equal to $R = \frac{\log |\mathcal{C}|}{c \log n \log q} = \frac{\log \frac{n}{c \log n}}{O_q(\log n)} = \Omega_q(1)$. Finally, since $|S(i \cdot c \log n, (i+1) \cdot c \log n]| = c \log n$, one can compute $S(i \cdot c \log n, (i+1) \cdot c \log n]$ in linear time in terms of its length. □

Our boosting step is mainly built on the converse of this observation.

Lemma 8.3.10. *Suppose $\mathcal{C}$ is a block insdel code over alphabet of size q, block length N, distance $1 - \varepsilon$ and rate R and let S be a string obtained by attaching all codewords back to back in any order. Then, for $\varepsilon' = 4\varepsilon$, S is a string of length $n = q^{R \cdot N} \cdot N$ which satisfies the long-distance ε'-synchronization property for any pair of intervals of aggregated length $\frac{4}{\varepsilon} N \leq \frac{4}{\varepsilon \log q}(\log n - \log R)$ or more. Further, if $\mathcal{C}$ is linear-time encodable, S has a highly explicit construction.*

Proof. The length of S follows from the definition of rate. Moreover, the highly explicitness follows from the fact that every substring of S of length $\log n$ may include parts of $\frac{1}{\varepsilon \log q} + 1$ codewords each of which can be computed in linear time in terms of their length. Therefore, any substring $S[i, i + \log n]$ can be constructed in $O\left(\max\left\{\frac{\log n}{\varepsilon \log q}, \log n\right\}\right) = O_{\varepsilon,q}(\log n)$. To prove the long distance property, we have to show that for every four indices $i_1 < j_1 \leq i_2 < j_2$ where $j_1 + j_2 - i_1 - i_2 \geq \frac{4N}{\varepsilon}$, we have

$$\mathrm{ED}(S[i_1, j_1), S[i_2, j_2)) \geq (1 - 4\varepsilon)(j_1 + j_2 - i_1 - i_2). \tag{8.9}$$

Assume that $S[i_1, j_1)$ contains a total of p complete blocks of $\mathcal{C}$ and $S[i_2, j_2)$ contains q complete blocks of $\mathcal{C}$. Let $S[i'_1, j'_1)$ and $S[i'_2, j'_2)$ be the strings obtained be throwing the partial blocks away from $S[i_1, j_1)$ and $S[i_2, j_2)$. Note that the overall length of the partial blocks in $S[i_1, j_1)$ and $S[i_2, j_2)$ is less than $4N$, which is at most an ε-fraction of $S[i_1, j_1) \cup S[i_2, j_2)$, since $\frac{4N}{4N/\varepsilon} < \varepsilon$.

Assume by contradiction that $\mathrm{ED}(S[i_1, j_1), S[i_2, j_2)) < (1-4\varepsilon)(j_1 + j_2 - i_1 - i_2)$. Since edit distance preserves the triangle inequality, we have that

$$\begin{aligned} ED\left(S[i'_1, j'_1), S[i'_2, j'_2)\right) &\leq ED\left(S[i_1, j_1), S[i_2, j_2)\right) + |S[i_1, i'_1)| + |S[j'_1, j_1)| + |S[i_2, i'_2)| + |S[j'_2, j_2)| \\ &\leq (1-4\varepsilon)\left(j_1 + j_2 - i_1 - i_2\right) + \varepsilon(j_1 + j_2 - i_1 - i_2) \\ &\leq (1-4\varepsilon+\varepsilon)\left(j_1 + j_2 - i_1 - i_2\right) \\ &< \left(\frac{1-3\varepsilon}{1-\varepsilon}\right)\left((j'_1 - i'_1) + (j'_2 - i'_2)\right). \end{aligned}$$

This means that the longest common subsequence of $S[i'_1, j'_1)$ and $S[i'_2, j'_2)$ has length of at least

$$\frac{1}{2}\left[\left(|S[i'_1, j'_1)| + |S[i'_2, j'_2)|\right)\left(1 - \frac{1-3\varepsilon}{1-\varepsilon}\right)\right],$$

which means that there exists a monotonically increasing matching between $S[i'_1, j'_1)$ and $S[i'_2, j'_2)$ of the same size. Since the matching is monotone, there can be at most $p+q$ pairs of error-correcting code blocks having edges to each other. The Pigeonhole Principle implies that there are two error-correcting code blocks B_1 and B_2 such that the number of edges between them is at least

$$\begin{aligned} & \frac{\frac{1}{2}\left[\left(|S[i_1, j_1)| + |S[i_2, j_2)|\right)\left(1 - \frac{1-3\varepsilon}{1-\varepsilon}\right)\right]}{p+q} \\ = \; & \frac{(p+q)N\left(1 - \frac{1-3\varepsilon}{1-\varepsilon}\right)}{2(p+q)} \\ > \; & \frac{1}{2}\left(1 - \frac{1-3\varepsilon}{1-\varepsilon}\right) \cdot N. \end{aligned}$$

Notice that this is also a lower bound on the longest common subsequence of B_1 and B_2. This means that

$$\mathrm{ED}(B_1, B_2) < 2N - \left(1 - \frac{1-3\varepsilon/4}{1-\varepsilon/4}\right)N < \left(1 + \frac{1-3\varepsilon}{1-\varepsilon}\right)N = \frac{2-4\varepsilon}{1-\varepsilon}N < 2\left(1-\varepsilon\right)N.$$

This contradicts the error-correcting code's distance property, which we assumed to be larger than $2(1-\varepsilon)N$, and therefore we may conclude that for all indices $i_1 < j_1 \leq i_2 < j_2$ where $j_1 + j_2 - i_1 - i_2 \geq \frac{4N}{\varepsilon}$, (8.9) holds. □

We point out that even a brute force enumeration of a good insdel code could be used to find a string that satisfies ε-synchronization property for pairs of intervals with large total length. All needed to get an ε-synchronization string is to concatenate that with a string which satisfies ε-synchronization property for small intervals. This one could be brute forced as well. Overall, this gives an alternative polynomial time construction (still using the inspiration of long-distance strings, though). More importantly, if we use a linear time construction for short intervals and a linear time encodable insdel code for long ones,

we get a simple $O_\varepsilon(\log n)$-explicit long-distance ε-synchronization string construction for which any interval $[i, i + O_\varepsilon(\log n)]$ is computable in $O_\varepsilon(\log n)$.

In the rest of this section, as depicted in Figure 8.1, we first introduce a high-distance, small-alphabet error correcting code that is encodable in linear time in Lemma 8.3.13 using a high-distance linear-time code introduced in [GI05]. We then turn this code into a high-distance insertion-deletion code using the indexing technique from Chapter 3. Finally, we will employ this insertion-deletion code in the setup of Lemma 8.3.10 to obtain a highly-explicit linear-time long-distance synchronization strings.

Our codes are based on the following code from Guruswami and Indyk [GI05].

Theorem 8.3.11 (Theorem 3 from [GI05])**.** *For every r, $0 < r < 1$, and all sufficiently small $\epsilon > 0$, there exists a family of codes of rate r and relative distance at least $(1 - r - \epsilon)$ over an alphabet of size $2^{O(\epsilon^{-4} r^{-1} \log(1/\epsilon))}$ such that codes from the family can be encoded in linear time and can also be (uniquely) decoded in linear time from $(1 - r - \epsilon)$ fraction of half-errors, i.e., a fraction e of errors and s of erasures provided $2e + s \leq (1 - r - \epsilon)$.*

One major downside of constructing ε-synchronization strings based on the code from Theorem 8.3.11 is the exponentially large alphabet size in terms of ε. We concatenate this code with an appropriate small alphabet code to obtain a high-distance code over a smaller alphabet size.

Lemma 8.3.12. *For sufficiently small ε and $A, R > 1$, and any set Σ_i of size $|\Sigma_i| = 2^{O(\varepsilon^{-5} \log(1/\varepsilon))}$, there exists a code $C : \Sigma_i \to \Sigma_o^N$ with distance $1 - \varepsilon$ and rate ε^R where $|\Sigma_o| = O(\varepsilon^{-A})$.*

Proof. To prove the existence of such code, we show that a random code with distance $\delta = 1 - \varepsilon$, rate $r = \varepsilon^A$, alphabet size $|\Sigma_o| = \varepsilon^{-A}$, and block length

$$N = \frac{\log |\Sigma_i|}{\log |\Sigma_o|} \cdot \frac{1}{r} = O\left(\frac{\varepsilon^{-5} \log(1/\varepsilon)}{A \log(1/\varepsilon)} \cdot \frac{1}{\varepsilon^R}\right) = \frac{1}{A} \cdot O\left(\varepsilon^{-5-R}\right)$$

exists with non-zero probability. The probability of two randomly selected codewords of length N out of Σ_o being closer than $\delta = 1 - \varepsilon$ can be bounded above by the following term.

$$\binom{N}{N\varepsilon} \left(\frac{1}{|\Sigma_o|}\right)^{-N\varepsilon}$$

Hence, the probability of the random code with $|\Sigma_o|^{Nr} = |\Sigma_1|$ codewords having a minimum distance smaller than $\delta = 1 - \varepsilon$ is at most the following.

$$\begin{aligned}
& \binom{N}{N\varepsilon} \left(\frac{1}{|\Sigma_o|}\right)^{N\varepsilon} \binom{|\Sigma_i|}{2} \\
\leq\ & \left(\frac{Ne}{N\varepsilon}\right)^{N\varepsilon} \frac{|\Sigma_i|^2}{|\Sigma_o|^{N\varepsilon}} \\
=\ & \left(\frac{e}{\varepsilon}\right)^{N\varepsilon} \frac{2^{O(\varepsilon^{-5} \log(1/\varepsilon))}}{\left(\varepsilon^{-A}\right)^{N\varepsilon}} \\
=\ & 2^{O((1-A) \log(1/\varepsilon) N\varepsilon + \varepsilon^{-5} \log(1/\varepsilon))} \\
=\ & 2^{(1-A) O(\varepsilon^{-4-R} \log(1/\varepsilon)) + O(\varepsilon^{-5} \log(1/\varepsilon))}
\end{aligned}$$

For $A > 1$, $1 - A$ is negative and for $R > 1$, $\varepsilon^{-4-R}\log(1/\varepsilon)$ is asymptotically larger than $\varepsilon^{-5}\log(1/\varepsilon)$. Therefore, for sufficiently small ε, the exponent is negative and the desired code exists. □

Concatenating the code from Theorem 8.3.11 (as the outer code) and the code from Lemma 8.3.12 (as inner code) gives the following code.

Lemma 8.3.13. *For sufficiently small ε and any constant $0 < \gamma$, there exists an error correcting code of rate $O(\varepsilon^{2.01})$ and distance $1-\varepsilon$ over an alphabet of size $O(\varepsilon^{-(1+\gamma)})$ which is encodable in linear time and also uniquely decodable from an e fraction of erasures and s fraction of symbol substitutions when $s + 2e < 1 - \varepsilon$ in linear time.*

Proof. To construct such code, we simply concatenate codes from Theorem 8.3.11 and Lemma 8.3.12 as outer and inner code respectively. Let $\mathcal{C}_1$ be an instantiation of the code from Theorem 8.3.11 with parameters $r = \varepsilon/4$ and $\epsilon = \varepsilon/4$. Code $\mathcal{C}_1$ is a code of rate $r_1 = \varepsilon/4$ and distance $\delta_1 = 1 - \varepsilon/4 - \varepsilon/4 = 1 - \varepsilon/2$ over an alphabet Σ_1 of size $2^{O(\epsilon^{-4}r^{-1}\log(1/\epsilon))} = 2^{O(\varepsilon^{-5}\log(1/\varepsilon))}$ which is encodable and decodable in linear time.

Further, according to Lemma 8.3.12, one can find a code $\mathcal{C}_2 : \Sigma_1 \to \Sigma_2^{N_2}$ for $\Sigma_2 = \varepsilon^{-(1+\gamma)}$ with distance $\delta_2 = 1 - \varepsilon/2$ rate $r_2 = O(\varepsilon^{1.01})$ by performing a brute-force search. Note that block length and alphabet size of $\mathcal{C}_2$ is constant in terms of n. Therefore, such code can be found in $O_\varepsilon(1)$ and by forming a look-up table can be encoded and decoded from δ half-errors in $O(1)$. Hence, concatenating codes $\mathcal{C}_1$ and $\mathcal{C}_2$ gives a code of distance $\delta = \delta_1 \cdot \delta_2 = (1 - \varepsilon/2)^2 \geq 1 - \varepsilon$ and rate $r = r_1 \cdot r_2 = O(\varepsilon^{2.01})$ over an alphabet of size $|\Sigma_2| = O\left(\varepsilon^{-(1+\gamma)}\right)$ which can be encoded in linear time in terms of block length and decoded from e fraction of erasures and s fraction of symbol substitutions when $s + 2e < 1 - \varepsilon$ in linear time as well. □

Indexing the codewords of a code from Lemma 8.3.13 with linear-time constructible synchronization strings of Theorem 8.3.7 using the technique from Chapter 3 summarized in Theorem 3.3.2 gives Theorem 8.3.14.

Theorem 8.3.14. *For sufficiently small ε, there exists a family of insertion-deletion codes with rate $\varepsilon^{O(1)}$ that correct from $1-\varepsilon$ fraction of insertions and deletions over an alphabet of size $\varepsilon^{O(1)}$ that is encodable in linear time and decodable in quadratic time in terms of the block length.*

Proof. Theorem 3.3.2 provides a technique to convert an error correcting code into an insertion-deletion code by indexing the codewords with a synchronization string. We use the error correcting code $\mathcal{C}$ from Lemma 8.3.13 with parameter $\varepsilon' = \varepsilon/2$ and $\gamma = 0.01$ along with a linear-time constructible synchronization strings S from Theorem 8.3.7 with parameter $\varepsilon'' = (\varepsilon/2)^2$ in the context of Theorem 3.3.2. We also use the global repositioning algorithm from Chapter 3 for the synchronization string (see Theorem 8.2.5). This will give an insertion-deletion code over an alphabet of size $\varepsilon^{O(1)}$ corrects from $(1-\varepsilon') - \sqrt{\varepsilon''} = 1-\varepsilon$ insdels with a rate of

$$\frac{r_{\mathcal{C}}}{1 + |\Sigma_S|/|\Sigma_{\mathcal{C}}|} = \frac{O\left(\varepsilon^{2.01}\right)}{1 + O(\varepsilon''^{-O(1)}/\varepsilon^{-1.01})} = \varepsilon^{O(1)}.$$

As $\mathcal{C}$ is encodable and S is constructible in linear time, the encoding time for the insdel code will be linear. Further, as $\mathcal{C}$ is decodable in linear time and S is decodable in quadratic time (using the global repositioning algorithm from Theorem 8.2.5), the code is decodable in quadratic time. □

Using insertion-deletion code from Theorem 8.3.14 and boosting step from Lemma 8.3.10, we can now proceed to the main theorem of this section that provides a highly explicit construction for $c = O_\varepsilon(1)$-long-distance synchronization strings.

Theorem 8.3.15. *There is a deterministic algorithm that, for any constant $0 < \varepsilon < 1$ and $n \in \mathbb{N}$, computes a $c = \varepsilon^{-O(1)}$-long-distance ε-synchronization string $S \in \Sigma^n$ where $|\Sigma| = \varepsilon^{-O(1)}$. Moreover, this construction is $O(\log n)$-explicit and can even compute $S[i, i+\log n]$ in $O_\varepsilon(\log n)$ time.*

Proof. We simply use an insertion-deletion code from Theorem 8.3.14 with parameter $\varepsilon' = \varepsilon/4$ and block length $N = \frac{\log_q n}{R}$ where $q = \varepsilon^{-O(1)}$ is the size of the alphabet from Theorem 8.3.14. Using this code in Lemma 8.3.10 gives a string S of length $q^{RN} \cdot N \geq n$ that satisfies $4\varepsilon' = \varepsilon$-synchronization property over any pair of intervals of total length $\frac{4N}{\varepsilon} = O\left(\frac{\log n}{\varepsilon R \log q}\right) = O\left(\varepsilon^{-O(1)} \log n\right)$ or more. Since the insertion-deletion code from Theorem 8.3.14 is linearly encodable, the construction will be highly-explicit.

To turn S into a c-long-distance ε-synchronization string for $c = \frac{4N}{\varepsilon \log n} = O\left(\varepsilon^{-O(1)}\right)$, we simply concatenate it with a string T that satisfies ε-synchronization property for neighboring intervals of total size smaller than $c \log n$. In other words, we propose the following structure for constructing c-long-distance ε-synchronization string R.

$$R[i] = (S[i], T[i]) = \left(\mathcal{C}\left(\left\lfloor \frac{i}{N} \right\rfloor\right)[i \,(\mathrm{mod}\, N)], T[i]\right) \tag{8.10}$$

Let S' be an ε-synchronization string of length $2c \log n$. Using linear-time construction from Theorem 8.3.7, one can find S' in linear time in its length, i.e, $O(\log n)$. We define strings T_1 and T_2 consisting of repetitions of S' as follows.

$$T_1 = (S', S', \cdots, S'), \qquad T_2 = (0^{c \log n}, S', S', \cdots, S')$$

The string $T_1 \cdot T_2$ satisfies ε-synchronization strings for neighboring intervals of total length $c \log n$ or less as any such substring falls into one copy of S'. Note that having S' one can find any symbol of T in linear time. Hence, T has a highly-explicit linear time construction. Therefore, concatenating S and T gives a linear time construction for c-long-distance ε-synchronization strings over an alphabet of size $\varepsilon^{-O(1)}$ that is highly-explicit and, further, allows computing any substring $[i, i + \log n]$ in $O(\log n)$ time. A schematic representation of this construction can be found in Figure 8.2. □

8.3.5 Infinite Synchronization Strings: Highly Explicit Construction

Throughout this section, we focus on construction of infinite synchronization strings. To measure the efficiency of a an infinite string's construction, we consider the required time

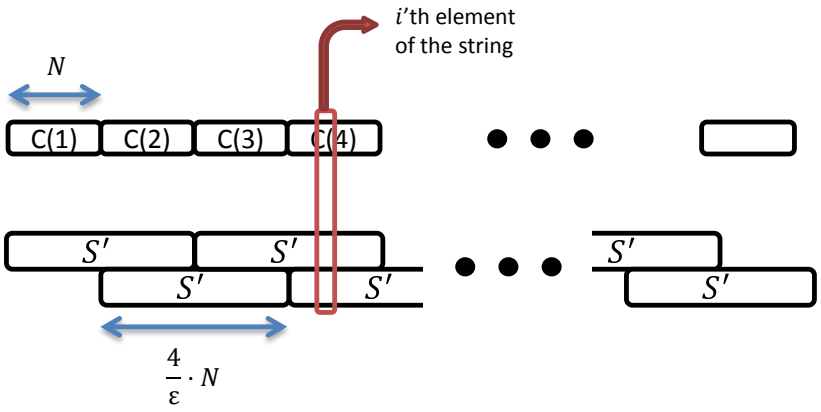


Figure 8.2: Pictorial representation of the construction of a long-distance ε-synchronization string of length n.

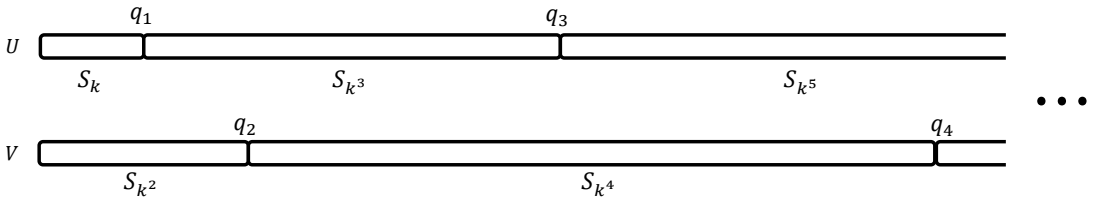


Figure 8.3: Construction of Infinite synchronization string T

complexity for computing the first n elements of that string. Moreover, besides the time complexity, we employ a generalized notion of explicitness to measure the quality of infinite string constructions.

In a similar fashion to finite strings, an infinite synchronization string is called to have a $T(n)$-*explicit* construction if there is an algorithm that computes any position $S[i]$ in $O\left(T(i)\right)$. Moreover, it is said to have a highly-explicit construction if $T(i) = O(\log i)$.

We show how to deterministically construct an infinitely-long ε-synchronization string over an alphabet Σ which is polynomially large in ε^{-1}. Our construction can compute the first n elements of the infinite string in $O(n)$ time, is highly-explicit, and, further, can compute any $[i, i + \log i]$ in $O(\log i)$.

Theorem 8.3.16. *For all $0 < \varepsilon < 1$, there exists an infinite ε-synchronization string construction over a poly(ε^{-1})-sized alphabet that is highly-explicit and also is able to compute $S[i, i + \log i]$ in $O(\log i)$. Consequently, using this construction, the first n symbols of the string can be computed in $O(n)$ time.*

Proof. Let $k = \frac{6}{\varepsilon}$ and let S_i denote a $\frac{\varepsilon}{2}$-synchronization string of length i. We define U and V as follows:

$$U = (S_k, S_{k^3}, S_{k^5}, \dots), \qquad V = (S_{k^2}, S_{k^4}, S_{k^6}, \dots)$$

In other words, U is the concatenation of $\frac{\varepsilon}{2}$-synchronization strings of length $k, k^3, k^5, \ldots$ and V is the concatenation of $\frac{\varepsilon}{2}$-synchronization strings of length $k^2, k^4, k^6, \ldots$. We build an infinite string T such that $T[i] = (U[i], V[i])$ (see Figure 8.3).

First, if finite synchronization strings S_{k^l} used above are constructed using the highly-explicit construction algorithm introduced in Theorem 8.3.15, any index i can be computed by simply finding one index in two of S_{k^l}s in $O(\log n)$. Further, any substring of length n of this construction can be computed by constructing finite synchronization strings of total length $O(n)$. According to Theorem 8.3.15, that can be done in $O_\varepsilon(n)$.

Now, all that remains is to show that T is an ε-synchronization string. We use following lemma to prove this.

Lemma 8.3.17. *Let $x < y < z$ be positive integers and let t be such that $k^t \leq |T[x, z)| < k^{t+1}$. Then there exists a block of S_{k^i} in U or V such that all but a $\frac{3}{k}$ fraction of $T[x, z)$ is covered by S_{k^i}.*

Note that this lemma shows that

$$\begin{aligned} \mathrm{ED}(T[x,y), T[y,z)) &> \left(1-\frac{\varepsilon}{2}\right)(|T[x,y)| + |T[y,z)|)\left(1-\frac{3}{k}\right) \\ &= \left(1-\frac{\varepsilon}{2}\right)^2(|T[x,y)| + |T[y,z)|) \geq (1-\varepsilon)(|T[x,y)| + |T[y,z)|) \end{aligned}$$

which implies that T is an ε-synchronization string. □

Proof of Lemma 8.3.17. We first define *i^{th} turning point* q_i to be the index of T at which $S_{k^{i+1}}$ starts, i.e., $q_i = k^i + k^{i-2} + k^{i-4} + \cdots$. Note that

$$q_i = \begin{cases} k^2 + k^4 + \cdots + k^i & \text{Even } i \\ k + k^3 + \cdots + k^i & \text{Odd } i \end{cases} \tag{8.11}$$

$$= \begin{cases} k^2 \frac{k^i - 1}{k^2 - 1} & \text{Even } i \\ k \frac{k^{i+1} - 1}{k^2 - 1} & \text{Odd } i \end{cases} \tag{8.12}$$

Note that $q_{t-1} < 2k^{t-1}$ and $|T[x, z)| \geq k^t$. Therefore, one can throw away all the elements of $T[x, z)$ whose indices are less than q_{t-1} without losing more than a $\frac{2}{k}$ fraction of the elements of $T[x, z)$. We will refer to the remaining part of $T[x, z)$ as $\tilde{T}$.

Now, the distance of any two turning points q_i and q_j where $t \leq i < j$ is at least $q_{t+1} - q_t$, and

$$q_{t+1} - q_t = \begin{cases} k \frac{k^{t+2} - 1}{k^2 - 1} - k^2 \frac{k^t - 1}{k^2 - 1} & \text{Even } t \\ k^2 \frac{k^{t+1} - 1}{k^2 - 1} - k \frac{k^{t+1} - 1}{k^2 - 1} & \text{Odd } t \end{cases} \tag{8.13}$$

$$= \begin{cases} \frac{(k-1)(k^{t+2}+k)}{k^2-1} = \frac{k^{t+2}+k}{k+1} & \text{Even } t \\ \frac{(k-1)(k^{t+2}-k)}{k^2-1} = \frac{k^{t+2}-k}{k+1} & \text{Odd } t. \end{cases} \tag{8.14}$$

Hence, $q_{t+1} - q_t > k^{t+1}\left(1 - \frac{1}{k}\right)$. Since $|\tilde{T}| \leq |T[x, z)| < k^{t+1}$, this fact gives that there exists a S_{k^i} which covers a $\left(1 - \frac{1}{k}\right)$ fraction of $\tilde{T}$. This completes the proof of the lemma. □

A similar discussion for infinite long-distance synchronization string can be found in Section 8.9.

8.4 Local Decoding

In Section 8.3, we discussed the close relationship between long-distance synchronization strings and insertion-deletion codes and provided highly-explicit constructions of long-distance synchronization strings based on insdel codes.

In this section, we make a slight modification to the highly explicit structure (8.10) we introduced in Theorem 8.3.15 where we showed one can use a constant rate insertion-deletion code $\mathcal{C}$ with distance $1-\frac{\varepsilon}{4}$ and block length $N = O(\log n)$ and a string T satisfying ε-synchronization property for pairs of neighboring intervals of total length $c \log n$ or less to make a c-long-distance synchronization string of length n. In addition to the symbols of the string consisting of codewords of $\mathcal{C}$ and symbols of string T, we append $\Theta\left(\log \frac{1}{\varepsilon}\right)$ extra bits to each symbol to enable *local decodability*. This extra symbol, as described in (8.15), essentially works as a circular index counter for insertion-deletion codewords.

$$R[i] = \left(\mathcal{C}\left(\left\lfloor \frac{i}{N} \right\rfloor \right) [i \pmod N], T[i], \left\lfloor \frac{i}{N} \right\rfloor \left(\mathrm{mod}\ \frac{8}{\varepsilon^3} \right) \right) \tag{8.15}$$

With this extra information appended to the construction, we claim that if the *relative suffix error density* is smaller than ε upon arrival of some symbol, then one can decode the corresponding index correctly by only looking at the last $O(\log n)$ symbols. At any point of a communication over an insertion-deletion channel, relative suffix error density is defined as the maximum fraction of errors occurred over all suffixes of the message sent so far. (Definition 5.12 from Definition 3.4.2).

Theorem 8.4.1. *Let R be a highly-explicit long-distance ε-synchronization string constructed according to* (8.15)*. Let $R[1,i]$ be sent by Alice and be received as $R'[1,j]$ by Bob. If relative suffix error density is smaller than $1-\frac{\varepsilon}{2}$, then Bob can find i in $\frac{4}{\varepsilon} \cdot T_{Dec}(N) + \frac{4N}{\varepsilon} \cdot (T_{Enc}(N) + Ex_T(c \log n) + c^2 \log^2 n)$ only by looking at the last $\max(\frac{4N}{\varepsilon^2}, c \log n)$ received symbols where T_{Enc} and T_{Dec} is the encoding and decoding complexities of $\mathcal{C}$ and $Ex_T(l)$ is the amount of time it takes to construct a substring of T of length l.*

For linear-time encodable, quadratic-time decodable code $\mathcal{C}$ and highly-explicit string T constructed by repetitions of short synchronization strings used in Theorem 8.3.15, construction (8.15) provides the following.

Theorem 8.4.2. *Let R be a highly-explicit long-distance ε-synchronization string constructed according to* (8.15) *with code $\mathcal{C}$ and string T as described in Theorem 8.3.15. Let $R[1,i]$ be sent by Alice and be received as $R'[1,j]$ by Bob. If relative suffix error density is smaller than $1-\frac{\varepsilon}{2}$, then Bob can find i in $O(\log^3 n)$ time only by looking at the last $O(\log n)$ received symbols.*

This decoding procedure, which we will refer to as *local decoding* consists of two principal phases upon arrival of each symbol. During the first phase, the receiver finds a list of $\frac{1}{\varepsilon}$ numbers that is guaranteed to contain the index of the actual codeword associated with the current position. This gives $\frac{N}{\varepsilon}$ candidates for the index of the received symbol. The second phase uses the relative suffix error density guarantee to choose the correct candidate among this list. The following lemma formally presents the first phase. This resembles a commonly utilized idea of using list-decoding as a middle step in the decoding procedure [GL16, GW17, GH14, GR06].

Lemma 8.4.3. *Let S be an ε-synchronization string constructed as described in* (8.15)*. Let $S[1,i]$ be sent by Alice and be received as $S_\tau[1,j]$ by Bob. If relative suffix error density is smaller than $1-\varepsilon/2$, then Bob can compute a list of $\frac{4N}{\varepsilon}$ numbers that is guaranteed to contain i.*

Proof. Note that as relative suffix error density is smaller than $1-\varepsilon/2 < 1$, the last received symbol has to be successfully transmitted. Therefore, Bob can correctly figure out the insertion-deletion code block index counter value which we denote by *count*. Note that if there are no errors, all symbols in blocks with index counter value of $count, count-1, \cdots, count-4/\varepsilon+1 \bmod \frac{8}{\varepsilon^3}$ that was sent by Bob right before the current symbol, have to be arrived within the past $4/\varepsilon \cdot N$ symbols. However, as adversary can insert symbols, those symbols can appear anywhere within the last $\frac{2}{\varepsilon}\frac{4N}{\varepsilon} = \frac{8N}{\varepsilon^2}$ symbols.

Hence, if Bob looks at the symbols arrived with index $i \in \{count, count-1, \cdots, count-4/\varepsilon+1\} \bmod \frac{8}{\varepsilon^3}$ within the last $\frac{8N}{\varepsilon^2}$ received symbols, he can observe all symbols coming from blocks with index $count, count-1, \cdots, count-4/\varepsilon+1 \bmod \frac{8}{\varepsilon^3}$ that was sent right before $S[i]$. Further, as our counter counts modulo $\frac{8}{\varepsilon^3}$, no symbols from older blocks with indices $count, count-1, \cdots, count-1/\varepsilon+1 \bmod \frac{8}{\varepsilon^3}$ will appear within the past $\frac{8N}{\varepsilon^2}$ symbols due to adversary's deletions. Therefore, Bob can find the symbols from the last $\frac{4}{\varepsilon}$ blocks up to some insdel errors. By decoding those blocks, he can make up a list of $\frac{4}{\varepsilon}$ candidates for the actual codeword block number associated with the received symbol. As each block contains N elements, there are a total of $\frac{4N}{\varepsilon}$ many candidates for i.

Note that as relative suffix error density is at most $1-\varepsilon/2$ and the last block may not have been completely sent yet, the total fraction of insdels in reconstruction of the last $\frac{4}{\varepsilon}$ blocks on Bob's side smaller than $1-\varepsilon/2+\frac{N}{4N/\varepsilon^2} \le 1-\frac{\varepsilon}{4}$. Therefore, the error density in at least one of those blocks is not larger than $1-\frac{\varepsilon}{4}$. This guarantees that at least one block will be correctly decoded and, therefore, the list contains the correct actual index. □

We now define a limited version of relative suffix distance which enables us to find the correct index among candidates found in Lemma 8.4.3.

Definition 8.4.4 (Limited Relative Suffix Distance)**.** *For any two strings $S, S' \in \Sigma^*$ we define their l-limited relative suffix distance, l-*LRSD*, as follows:*

$$l\text{-LRSD}(S,S') = \max_{0<k<l} \frac{\text{ED}\left(S(|S|-k,|S|], S'(|S'|-k,|S'|]\right)}{2k}$$

Note that $l = O(\log n)$-limited suffix distance of two strings can be computed in $O(l^2) = O(\log^2 n)$ by computing edit distance of all pairs of prefixes of their l-long suffixes.

Lemma 8.4.5. *If string S is a c-long distance ε-synchronization string, then for any two distinct prefixes $S[1,i]$ and $S[1,j]$, $(c\log n)$-LRSD$(S[1,i], S[1,j]) > 1-\varepsilon$.*

Proof. If $j - i < c\log n$, the synchronization string property gives that ED$(S(2i - j, i], S(i,j]) > 2(j-i)(1-\varepsilon)$ which gives the claim for $k = j - i$. If $j - i \geq c\log n$, the long-distance property gives that ED$(S(i - \log n, i], S(j - \log n, j]) > 2(1-\varepsilon)c\log n$ which again, proves the claim. □

Lemmas 8.4.3 and 8.4.5 enable us to prove Theorem 8.4.1.

Proof of Theorem 8.4.1. Using Lemma 8.4.3, by decoding $4/\varepsilon$ codewords, Bob forms a list of $4N/\varepsilon$ candidates for the index of the received symbol. This will take $4/\varepsilon \cdot T_{Dec}(N)$ time. Then, using Lemma 8.4.5, for any of the $4N/\varepsilon$ candidates, he has to construct a $c\log n$ substring of R and compute the $(c\log n)$-LRSD of that with the string he received. This requires looking at the last $\max(4n/\varepsilon, c\log n)$ recieved symbols and takes $4N/\varepsilon \cdot (T_{Enc}(N) + Ex_T(c\log n) + c^2\log^2 n)$ time. □

8.5 Application: Near Linear Time Codes Against Insdels, Block Transpositions, and Block Replications

In Sections 8.3 and 8.4, we provided highly explicit constructions and local repositionings for synchronization strings. Utilizing these two important properties of synchronization strings together suggests important improvements over insertion-deletion codes introduced in Chapter 3. We start by stating the following important lemma which summarizes the results of Sections 8.3 and 8.4.

Lemma 8.5.1. *For any $0 < \varepsilon < 1$, there exists an streaming (n,δ)-indexing solution with ε-synchronization string S and streaming decoding algorithm $\mathcal{D}$ that figures out the index of each symbol by merely considering the last $O_\varepsilon(\log n)$ received symbols and in $O_\varepsilon(\log^3 n)$ time. Further, $S \in \Sigma^n$ is highly-explicit and constructible in linear-time and $|\Sigma| = O\left(\varepsilon^{-O(1)}\right)$. This solution may contain up to $\frac{n\delta}{1-\varepsilon}$ misdecodings.*

Proof. Let S be a long-distance 2ε-synchronization string constructed according to Theorem 8.3.15 and enhanced as suggested in (8.15) to ensure local decodablity. As discussed in Sections 8.3 and 8.4, these strings trivially satisfy all properties claimed in the statement other than the misdecoding guarantee.

According to Theorem 8.4.2, correct decoding is ensured whenever relative suffix error density is less than $1 - \frac{2\varepsilon}{2} = 1 - \varepsilon$. Therefore, as relative suffix error density can exceed $1-\varepsilon$ upon arrival of at most $\frac{n\delta}{1-\varepsilon}$ many symbols (see Lemma 3.4.14), there can be at most $\frac{n\delta}{1-\varepsilon}$ many successfully received symbols which are not decoded correctly. This proves the misdecoding guarantee. □

8.5.1 Near-Linear Time Insertion-Deletion Code

Using the indexing technique from Theorem 3.3.2 with synchronization strings and decoding algorithm from Theorem 8.2.5, one can obtain the following insdel codes.

Theorem 8.5.2. *For any $0 < \delta < 1/3$ and sufficiently small $\varepsilon > 0$, there exists an encoding map $E : \Sigma^k \to \Sigma^n$ and a decoding map $D : \Sigma^* \to \Sigma^k$, such that, if $EditDistance(E(m), x) \leq \delta n$ then $D(x) = m$. Further, $\frac{k}{n} > 1 - 3\delta - \varepsilon$, $|\Sigma| = f(\varepsilon)$, and E and D can be computed in $O(n)$ and $O(n \log^3 n)$ time respectively.*

Proof. We closely follow the proof of Theorem 3.1.1 and use Theorem 3.3.2 to convert a near-MDS error correcting code to an insertion-deletion code satisfying the claimed properties.

Given the δ and ε, we choose $\varepsilon' = \frac{\varepsilon}{12}$ and use locally decodable $O_{\varepsilon'}(1)$-long-distance ε'-synchronization string S of length n over alphabet Σ_S of size $\varepsilon'^{-O(1)} = \varepsilon^{-O(1)}$ from Theorem 8.4.2. We plug this synchronization string with the local decoding from Theorem 8.4.2 into Theorem 3.3.2 with a near-MDS expander code [GI05] $\mathcal{C}$ (see Theorem 8.3.11) which can efficiently correct up to $\delta_{\mathcal{C}} = 3\delta + \frac{\varepsilon}{3}$ half-errors and has a rate of $R_{\mathcal{C}} > 1 - \delta_{\mathcal{C}} - \frac{\varepsilon}{3}$ over an alphabet $\Sigma_{\mathcal{C}}$ of size $\exp(\varepsilon^{-O(1)})$ such that $\log |\Sigma_{\mathcal{C}}| \geq \frac{3 \log |\Sigma_S|}{\varepsilon}$. This ensures that the final rate is indeed at least

$$\frac{R_{\mathcal{C}}}{1 + \frac{\log |\Sigma_S|}{\log |\Sigma_{\mathcal{C}}|}} \geq R_{\mathcal{C}} - \frac{\log |\Sigma_S|}{\log |\Sigma_{\mathcal{C}}|} = 1 - 3\delta - 3\frac{\varepsilon}{3} = 1 - 3\delta - \varepsilon$$

and the fraction of insdel errors that can be efficiently corrected is $\delta_{\mathcal{C}} - 2\frac{\delta}{1-\varepsilon'} \geq 3\delta + \varepsilon/3 - 2\delta(1 + 2\varepsilon') \geq \delta$. The encoding and decoding complexities are straightforward according to guarantees stated in Theorem 8.5.1 and the linear time construction of S. □

8.5.2 Insdels, Block Transpositions, and Block Replications

In this section, we introduce block transposition and block replication errors and show that code from Theorem 8.5.2 can overcome these types errors as well.

One can think of several way to model transpositions and replications of blocks of data. One possible model would be to have the string of data split into blocks of length l and then define transpositions and replications over those fixed blocks. In other words, for message $m_1, m_2, \cdots, m_n \in \Sigma^n$, a single transposition or replication would be defined as picking a block of length l and then move or copy that blocks of data somewhere in the message.

Another (more general) model is to let adversary choose any block, i.e., substring of the message he wishes and then move or copy that block somewhere in the string. Note that in this model, a constant fraction of block replications may make the message length exponentially large in terms of initial message length. We will focus on this more general model and provide codes protecting against them running near-linear time in terms of the length of the *received* message. Such results automatically extend to the weaker model that does not lead to exponentially large corrupted messages.

We now formally define (i, j, l)-*block transposition* as follows.

Definition 8.5.3 ((i,j,l)-Block Transposition)**.** *For a given string $M = m_1 \cdots m_n$, the* (i,j,l)-block transposition *operation for $1 \leq i \leq i+l \leq n$ and $j \in \{1, \cdots, i-1, i+l+1, \cdots, n\}$ is defined as an operation which turns M into*

$$M' = m_1, \cdots, m_{i-1}, m_{i+l+1} \cdots, m_j, m_i \cdots m_{i+l}, m_{j+1}, \cdots, m_n \text{ if } j > i+l$$

or

$$M' = m_1, \cdots, m_j, m_i, \cdots, m_{i+l}, m_{j+1}, \cdots, m_{i-1}, m_{i+l+1} \cdots, m_n \text{ if } j < i$$

by removing $M[i, i+l]$ and inserting it right after $M[j]$.

Also, *(i,j,l)-block replication* is defined as follows.

Definition 8.5.4 ((i,j,l)-Block Replication)**.** *For a given string $M = m_1 \cdots m_n$, the* (i,j,l)-block replication *operation for $1 \leq i \leq i+l \leq n$ and $j \in \{1, \cdots, n\}$ is defined as an operation which turns M into $M' = m_1, \cdots, m_j, m_i \cdots m_{i+l}, m_{j+1}, \cdots, m_n$ which is obtained by copying $M[i, i+l]$ right after $M[j]$.*

We now proceed to the following theorem that implies the code from Theorem 8.5.2 recovers from block transpositions and replications as well.

Theorem 8.5.5. *Let $S \in \Sigma_S^n$ be a locally-decodable highly-explicit c-long-distance ε-synchronization string from Theorem 8.4.2 and $\mathcal{C}$ be an half-error correcting code of block length n, alphabet $\Sigma_{\mathcal{C}}$, rate r, and distance d with encoding function $\mathcal{E}_{\mathcal{C}}$ and decoding function $\mathcal{D}_{\mathcal{C}}$ that run in $T_{\mathcal{E}_{\mathcal{C}}}$ and $T_{\mathcal{D}_{\mathcal{C}}}$ respectively. Then, one can obtain an encoding function $E_n : \Sigma_{\mathcal{C}}^{nr} \to [\Sigma_{\mathcal{C}} \times \Sigma_S]^n$ that runs in $T_{\mathcal{E}_{\mathcal{C}}} + O(n)$ and decoding function $D_n : [\Sigma_{\mathcal{C}} \times \Sigma_S]^* \to \Sigma_{\mathcal{C}}^{nr}$ which runs in $T_{\mathcal{D}_{\mathcal{C}}} + O\left(\log^3 n\right)$ and recovers from $n\delta_{insdel}$ fraction of synchronization errors and δ_{block} fraction of block transpositions or replications as long as $\left(2 + \frac{2}{1-\varepsilon/2}\right)\delta_{insdel} + (12c \log n)\delta_{block} < d$.*

Proof. To obtain such codes, we simply index the symbols of the given error correcting code with the symbols of the given synchronization strings. More formally, the encoding function $\mathcal{E}(x)$ for $x \in \Sigma_{\mathcal{C}}^{nr}$ first computes $\mathcal{E}_{\mathcal{C}}(x)$ and then indexes it, symbol by symbol, with the elements of the given synchronization string.

On the decoding end, $\mathcal{D}(x)$ first uses the indices on each symbol to guess the actual position of the symbols using the local decoding of the c-long-distance ε-synchronization string. Rearranging the received symbols in accordance to the guessed indices, the receiving end obtains a version of $\mathcal{E}_{\mathcal{C}}(x)$, denoted by $\bar{x}$, that may suffer from a number of symbol corruption errors due to incorrect index misdecodings. As long as the number of such misdecodings, k, satisfies $n\delta_{insdel} + 2k \leq nd$, computing $\mathcal{D}_{\mathcal{C}}(\bar{x})$ gives x. The decoding procedure naturally consists of decoding the attached synchronization string, rearranging the indices, and running $\mathcal{D}_{\mathcal{C}}$ on the rearranged version. Note that if multiple symbols where detected to be located at the same position by the synchronization string decoding procedure or no symbols where detected to be at some position, the decoder can simply put a special symbol '?' there and treat it as a half-error. The decoding and encoding complexities are trivial.

In order to find the actual index of a received symbol correctly, we need the local decoding procedure to compute the index correctly. For that purpose, it suffices that no block operations cut or paste symbols within an interval of length $2c\log n$ before that index throughout the entire block transpositions/replications performed by the adversary and the relative suffix error density caused by synchronization errors for that symbol does not exceed $1-\varepsilon/2$. As any block operation might cause three new cut/cop/paste edges and relative suffix error density is larger than $1-\varepsilon/2$ for up to $\frac{1}{1-\varepsilon/2}$ many symbols (according to Lemma 3.4.14), the positions of all but at most $k \leq 3n\delta_{block} \times 2c\log n + n\delta_{insdel}\left(1+\frac{1}{1-2\varepsilon}\right)$ symbols will be decoded incorrectly via synchronization string decoding procedure. Hence, as long as $n\delta_{insdel} + 2k \leq 6\delta_{block} \times 2c\log n + n\delta_{insdel}\left(3+\frac{2}{1-2\varepsilon}\right) < d$ the decoding procedure succeeds. Finally, the encoding and decoding complexities follow from the fact that indexing codewords of length n takes linear time and the local decoding of synchronization strings takes $O(n\log^3 n)$ time. □

Employing locally-decodable $O_\varepsilon(1)$-long-distance synchronization strings of Theorem 8.4.2 and error correcting code of Theorem 8.3.11 in Theorem 8.5.5 gives the following code.

Theorem 8.5.6. *For any $0 < r < 1$ and sufficiently small ε there exists a code with rate r that corrects $n\delta_{insdel}$ synchronization errors and $n\delta_{block}$ block transpositions or replications as long as $6\delta_{insdel} + c\log n\delta_{block} < 1 - r - \varepsilon$ for some $c = O(1)$. The code is over an alphabet of size $O_\varepsilon(1)$ and has $O(n)$ encoding and $O(N\log^3 n)$ decoding complexities where N is the length of the received message.*

8.6 Applications: Near-Linear Time Infinite Channel Simulations with Optimal Memory Consumption

We now show that the indexing algorithm introduced in Theorem 8.5.1 can improve the efficiency of channel simulations from Chapter 7 as well as insdel codes. Consider a scenario where two parties are maintaining a communication that suffers from synchronization errors, i.e, insertions and deletions. In Chapter 7, we provided a simple technique to overcome this desynchronization. Our solution consists of a simple symbol by symbol attachment of a synchronization string to any transmitted symbol. The attached indices enables the receiver to correctly detect indices of most of the symbols he receives. However, the decoding procedure introduced in Chapter 7 takes polynomial time in terms of the communication length. The explicit construction introduced in Section 8.3 and local repositioning provided in Section 8.4 can reduce the construction and decoding time and space complexities to polylogarithmic. Further, the decoding procedure only requires to look up $O_\varepsilon(\log n)$ recently received symbols upon arrival of any symbol.

Interestingly, we will show that, beyond the time and space complexity improvements over simulations in Chapter 7, long-distance synchronization strings can make *infinite channel simulations* possible. In other words, two parties communicating over an insertion-deletion channel are able to simulate a corruption channel on top of the given channel even if they are not aware of the length of the communication before it ends with similar

guarantees as of Chapter 7. To this end, we introduce infinite strings that can be used to index communications to convert synchronization errors into symbol corruptions. The following theorem analogous to the indexing algorithm of Lemma 8.5.1 provides all we need to perform such simulations.

Theorem 8.6.1. *For any $0 < \varepsilon < 1$, there exists an infinite string S that satisfies the following properties:*

1. *String S is over an alphabet of size $\varepsilon^{-O(1)}$.*
2. *String S has a highly-explicit construction and, for any i, $S[i, i+\log i]$ can be computed in $O(\log i)$.*
3. *Assume that $S[1, i]$ is sent over an insertion-deletion channel. There exists a decoding algorithm for the receiving side that, if relative suffix error density is smaller than $1 - \varepsilon$, can correctly find i by looking at the last $O(\log i)$ and knowing the number of received symbols in $O(\log^3 i)$ time.*

Proof. To construct such a string S, we use our finite-length highly-explicit locally-decodable long-distance synchronization string constructions from Theorem 8.4.2 and use to construct finite substrings of S as proposed in the infinite string construction of Theorem 8.3.16 which is depicted in Figure 8.3. We choose length progression parameter $k = 10/\varepsilon^2$. Similar to the proof of Lemma 8.3.17, we define *turning point* q_i as the index at which $S_{k^{i+1}}$ starts. We append one extra bit to each symbol $S[i]$ which is zero if $q_j \leq i < q_{j+1}$ for some even j and one otherwise.

This construction clearly satisfies the first two properties claimed in the theorem statement. To prove the third property, suppose that $S[1, i]$ is sent and received as $S'[1, i']$ and the error suffix density is less than $1 - \varepsilon$. As error suffix density is smaller than $1 - \varepsilon$, $i\varepsilon \leq i' \leq i/\varepsilon$ which implies that $i'\varepsilon \leq i \leq i'/\varepsilon$. This gives an uncertainty interval whose ends are close by a factor of $1/\varepsilon^2$. By the choice of k, this interval contains at most one turning point. Therefore, using the extra appended bit, receiver can figure out index j for which $q_j \leq i < q_{j+1}$. Knowing this, it can simply use the local decoding algorithm for finite string S_{j-1} to find i. □

Theorem 8.6.2. *[Improving Channel Simulations of Theorem 7.1.1]*

(a) Suppose that n rounds of a one-way/interactive insdel channel over an alphabet Σ with a δ fraction of insertions and deletions are given. Using an ε-synchronization string over alphabet Σ_{syn}, it is possible to simulate $n\,(1 - O_\varepsilon(\delta))$ rounds of a one-way/interactive corruption channel over Σ_{sim} with at most $O_\varepsilon\,(n\delta)$ symbols corrupted so long as $|\Sigma_{sim}| \times |\Sigma_{syn}| \leq |\Sigma|$.

(b) Suppose that n rounds of a binary one-way/interactive insertion-deletion channel with a δ fraction of insertions and deletions are given. It is possible to simulate $n(1 - \Theta(\sqrt{\delta \log(1/\delta)}))$ rounds of a binary one-way/interactive corruption channel with $\Theta(\sqrt{\delta \log(1/\delta)})$ fraction of corruption errors between two parties over the given channel.

Having an explicitly-constructible, locally-decodable, infinite string from Theorem 8.6.1 utilized in the simulation, all of the simulations mentioned above take $O(\log n)$ time for sending/starting party of one-way/interactive communications. Further, on the other side, the simulation spends $O(\log^3 n)$ time upon arrival of each symbol and only looks up $O(\log n)$ many recently received symbols. Overall, these simulations take a $O(n \log^3 n)$ time and $O(\log n)$ space to run. These simulations can be performed even if parties are not aware of the communication length.

Proof. We simply replace ordinary ε-synchronization strings used in all such simulations in Chapter 7 with the highly-explicit locally-decodable infinite string from Theorem 8.6.1 with its corresponding local-repositioning procedure instead of the minimum RSD decoding procedure that is used in Chapter 7. This keeps all properties that simulations from Chapter 7 guarantee. Further, by properties stated in Theorem 8.6.1, the simulation is performed in near-linear time, i.e., $O(n \log^3 n)$. Also, constructing and decoding each symbol of the string from Theorem 8.6.1 only takes $O(\log n)$ space which leads to an $O(\log n)$ memory requirement on both sides. □

8.7 Applications: Near-Linear Time Coding Scheme for Interactive Communication

Using the near-linear time interactive channel simulation in Theorem 8.6.2 with the near-linear time interactive coding scheme of Haeupler and Ghaffari [GH14] (stated in Theorem 8.7.1) gives the near-linear time coding scheme for interactive communication over insertion-deletion channels stated in Theorem 8.7.2.

Theorem 8.7.1 (Theorem 1.1 from [GH14])**.** *For any constant $\varepsilon > 0$ and n-round protocol Π there is a randomized non-adaptive coding scheme that robustly simulates Π against an adversarial error rate of $\rho \leq 1/4 - \varepsilon$ using $N = O(n)$ rounds, a near-linear $n \log^{O(1)} n$ computational complexity, and failure probability $2^{-\Theta(n)}$.*

Theorem 8.7.2. *For a sufficiently small δ and n-round alternating protocol Π, there is a randomized coding scheme simulating Π in presence of δ fraction of edit-corruptions with constant rate (i.e., in $O(n)$ rounds) and in near-linear time. This coding scheme works with probability $1 - 2^{\Theta(n)}$.*

8.8 Alphabet Size vs Distance Function

In this section, we study the dependence of alphabet size over distance function, f, for $f(l)$-distance synchronization strings. We will discuss this dependence for polynomial, exponential, and super exponential function f. As briefly mentioned in Section 8.3.1, we will show that for any polynomial function f, one can find arbitrarily long $f(l)$-distance ε-synchronization strings over an alphabet that is polynomially large in terms of ε^{-1} (Theorem 8.8.1). Also, in Theorem 8.8.2, we will show that one cannot hope for such guarantee over alphabets with sub-polynomial size in terms of ε^{-1}. Further, for exponential distance

function f, we will show that arbitrarily long $f(l)$-distance ε-synchronization strings exist over alphabets that are exponentially large in terms of ε^{-1} (Theorem 8.8.1) and, furthermore, cannot hope for such strings over alphabets with sub-exponential size in terms of ε^{-1} (Theorem 8.8.3). Finally, in Theorem 8.8.4, we will show that for super-exponential f, $f(l)$-distance ε-synchronization string does not exist over constant-sized alphabets in terms of string length.

Theorem 8.8.1. *For any polynomial function f, there exists an alphabet of size $O(\varepsilon^{-4})$ over which arbitrarily long $f(l)$-distance ε-synchronization strings exist. Further, for any exponential function f, such strings exist over an alphabet of size $\exp(\varepsilon^{-1})$.*

Proof. To prove this we follow the same LLL argument as in Theorem 8.3.5 and Chapter 3 to prove the existence of a string that satisfies the $f(l)$-distance ε-synchronization string property for intervals of length t or more and then concatenate it with $1, 2, \cdots, t, 1, 2, \cdots, t, \cdots$ to take care of short intervals. We define bad events B_{i_1,l_1,i_2,l_2} in the same manner as in Theorem 8.3.5 and follow similar steps up until (8.3) by proposing $x_{i_1,l_1,i_2,l_2} = D^{-\varepsilon(l_1+l_2)}$ for some $D > 1$ to be determined later. D has to be chosen such that for any i_1, l_1, i_2, l_2 and $l = l_1 + l_2$:

$$\left(\frac{e}{\varepsilon\sqrt{|\Sigma|}}\right)^{\varepsilon l} \leq D^{-\varepsilon l} \prod_{[S[i_1,i_1+l_1)\cup S[i_2,i_2+l_2)]\cap[S[i'_1,i'_1+l'_1)\cup S[i'_2,i'_2+l'_2)]\neq\emptyset} \left(1 - D^{-\varepsilon(l'_1+l'_2)}\right) \tag{8.16}$$

Note that:

$$D^{-\varepsilon l} \prod_{[S[i_1,i_1+l_1)\cup S[i_2,i_2+l_2)]\cap[S[i'_1,i'_1+l'_1)\cup S[i'_2,i'_2+l'_2)]\neq\emptyset} \left(1 - D^{-\varepsilon(l'_1+l'_2)}\right) \tag{8.17}$$

$$\geq D^{-\varepsilon l} \prod_{l'=t}^{n} \prod_{l'_1=1}^{l'} \left(1 - D^{-\varepsilon l'}\right)^{\left[(l_1+l'_1)+(l_1+l'_2)+(l_2+l'_1)+(l_2+l'_2)\right]f(l')} \tag{8.18}$$

$$= D^{-\varepsilon l} \prod_{l'=t}^{n} \left(1 - D^{-\varepsilon l'}\right)^{4l'(l+l')f(l')} \tag{8.19}$$

$$= D^{-\varepsilon l} \left[\prod_{l'=t}^{n} \left(1 - D^{-\varepsilon l'}\right)^{4l'f(l')}\right]^{l} \times \prod_{l'=t}^{n} \left(1 - D^{-\varepsilon l'}\right)^{4l'^2 f(l')} \tag{8.20}$$

$$\geq D^{-\varepsilon l} \left[1 - \sum_{l'=t}^{n} 4l'f(l')D^{-\varepsilon l'}\right]^{l} \times \left(1 - \sum_{l'=t}^{n} 4l'^2 f(l')D^{-\varepsilon l'}\right) \tag{8.21}$$

To bound below this term we use an upper-bound for series $\Sigma_{i=t}^{\infty} g(i)x^i$. Note that the proportion of two consecutive terms in such summation is at most $\frac{g(t+1)x^{t+1}}{g(t)x^t}$. Therefore,

$\Sigma_{i=t}^{\infty} g(i)x^i \leq \frac{g(t)x^t}{1-\frac{g(t+1)x^{t+1}}{g(t)x^t}}$. Therefore, for LLL to work, it suffices to have the following.

$$\left(\frac{e}{\varepsilon\sqrt{|\Sigma|}}\right)^{\varepsilon l} \leq D^{-\varepsilon l}\left[1-\frac{4tf(t)D^{-\varepsilon t}}{1-\frac{4tf(t+1)D^{-\varepsilon(t+1)}}{4tf(t)D^{-\varepsilon t}}}\right]^l \times \left(1-\frac{4t^2f(t)D^{-\varepsilon t}}{1-\frac{4t^2f(t+1)D^{-\varepsilon(t+1)}}{4t^2f(t)D^{-\varepsilon t}}}\right) \quad (8.22)$$

$$= D^{-\varepsilon l}\left[1-\frac{4tf(t)D^{-\varepsilon t}}{1-\frac{f(t+1)D^{-\varepsilon}}{f(t)}}\right]^l \times \left(1-\frac{4t^2f(t)D^{-\varepsilon t}}{1-\frac{f(t+1)D^{-\varepsilon}}{f(t)}}\right) \quad (8.23)$$

Polynomial Distance Function: For polynomial function $f(l) = \sum_{i=0}^{d} a_i l^i$ of degree d, we choose $t = 1/\varepsilon^2$ and $D = e$. This choice gives that

$$L_1 = \frac{4tf(t)D^{-\varepsilon t}}{1-\frac{f(t+1)D^{-\varepsilon}}{f(t)}} = \frac{4\varepsilon^{-2}f(\varepsilon^{-2})e^{-1/\varepsilon}}{1-(1+\varepsilon^2)^d e^{-\varepsilon}}$$

and

$$L_2 = \frac{4t^2f(t)D^{-\varepsilon t}}{1-\frac{f(t+1)D^{-\varepsilon}}{f(t)}} = \frac{4\varepsilon^{-4}f(\varepsilon^{-2})e^{-1/\varepsilon}}{1-(1+\varepsilon^2)^d e^{-\varepsilon}}.$$

We study the following terms in $\varepsilon \to 0$ regime. Note that $4\varepsilon^{-2}$ and $4\varepsilon^{-4}$ are polynomials in ε^{-1} but $e^{-1/\varepsilon}$ is exponential in ε^{-1}. Therefore, for sufficiently small ε,

$$4\varepsilon^{-2}f(\varepsilon^{-2})e^{-1/\varepsilon}, 4\varepsilon^{-4}f(\varepsilon^{-2})e^{-1/\varepsilon} \leq e^{-0.9/\varepsilon}.$$

Also, $1-(1+\varepsilon^2)^d e^{-\varepsilon} \leq 1-(1+\varepsilon^2)^d(1-\varepsilon/2) = 1-(1-\varepsilon/2+o(\varepsilon^2))$. So, for small enough ε, $1-(1+\varepsilon^2)^d e^{-\varepsilon} \leq \frac{3}{4}\varepsilon$. This gives that, for small enough ε,

$$L_1, L_2 \leq \frac{e^{-0.9/\varepsilon}}{(3/4)\varepsilon} \leq e^{-0.8/\varepsilon}. \quad (8.24)$$

Note that $1-e^{-0.8/\varepsilon} \geq e^{-\varepsilon}$ for $0 < \varepsilon < 1$. Plugging this fact into (8.23) gives that, for small enough ε, the LLL condition is satisfied if

$$\left(\frac{e}{\varepsilon\sqrt{|\Sigma|}}\right)^{\varepsilon l} \leq e^{-\varepsilon l}\cdot e^{-\varepsilon l}\cdot e^{-\varepsilon} \Leftrightarrow \left(\frac{e^3}{\varepsilon\sqrt{|\Sigma|}}\right)^{\varepsilon l} \leq \frac{1}{e^{\varepsilon}} \Leftrightarrow |\Sigma| \geq \frac{e^{6+2/l}}{\varepsilon^2} \Leftarrow |\Sigma| \geq \frac{e^8}{\varepsilon^2} = O(\varepsilon^{-2})$$

Therefore, for any polynomial f, $f(l)$-distance ε-synchronization strings exist over alphabets of size $t \times |\Sigma| = O(\varepsilon^{-4})$.

Exponential Distance Function: For exponential function $f(l) = c^l$, we choose $t = 1$ and $D = (8c)^{1/\varepsilon}$. Plugging this choice of t and D into (8.23) turns it into the following.

$$\left(\frac{e}{\varepsilon\sqrt{|\Sigma|}}\right)^{\varepsilon l} \leq D^{-\varepsilon l}\left[1-\frac{4tf(t)D^{-\varepsilon t}}{1-\frac{f(t+1)D^{-\varepsilon}}{f(t)}}\right]^l \times \left(1-\frac{4t^2f(t)D^{-\varepsilon t}}{1-\frac{f(t+1)D^{-\varepsilon}}{f(t)}}\right) \quad (8.25)$$

$$= (2c)^{-l}\left[1-\frac{4c(8c)^{-1}}{1-c^{\frac{1}{8c}}}\right]^l \times \left(1-\frac{4c(8c)^{-1}}{1-c^{\frac{1}{8c}}}\right) \quad (8.26)$$

$$= \frac{1}{(2c)^l}\cdot\left[1-\frac{1/2}{7/8}\right]^{l+1} = \frac{2\cdot(3/14)^{l+1}}{c^l} \quad (8.27)$$

Therefore, if $|\Sigma|$ satisfies the following, the LLL condition will be satisfied.

$$\left(\frac{e}{\varepsilon\sqrt{|\Sigma|}}\right)^{\varepsilon l} \leq \frac{2\cdot(3/14)^{l+1}}{c^l} \Leftarrow |\Sigma| \geq \frac{e^2}{\varepsilon^2}\cdot\left(\frac{14^2 c}{3^2}\right)^{2/\varepsilon}$$

Therefore, for any exponential f, $f(l)$-distance ε-synchronization strings exist over alphabets of size $c_0^{1/\varepsilon}$ where c_0 is a constant depending on the basis of the exponential function f. □

Theorem 8.8.2. *Any alphabet Σ over which arbitrarily long $f(l)$-distance ε-synchronization strings exist has to be of size $\Omega(\varepsilon^{-1})$. This holds for any function f.*

Proof. We simply prove this theorem for $f(l) = 0$, i.e., ordinary synchronization strings which trivially extends to general f. Note that ε-synchronization guarantee for any pair of intervals $[i, j)$ and $[j, k)$ where $k - i < \varepsilon^{-1}$ dictates that no symbol have to appear more than once in $[i, k)$. Therefore, the alphabet size has to be at least $\varepsilon^{-1} - 1$. □

Theorem 8.8.3. *Let f be an exponential function. If arbitrarily long $f(l)$-distance ε-synchronization strings exist over an alphabet Σ, the size of Σ has to be at least exponentially large in terms of ε^{-1}.*

Proof. Let $f(l) = c^l$. In a given $f(l)$-distance ε-synchronization string, take two intervals of length l_1 and l_2 where $l_1 + l_2 \leq \varepsilon^{-1}/2 < \varepsilon^{-1}$. The edit distance requirement of ε-synchronization definition requires those two intervals not to contain any similar symbols. Note that this holds for any two intervals of total length $l = \varepsilon^{-1}/2$ in a prefix of length $c^l = c^{\varepsilon^{-1}/2}$. Therefore, no symbol can be appear more than once throughout the first $c^{\varepsilon^{-1}/2}$ symbols of the given strings. This shows that the alphabet size has to be at least exponentially large in terms of ε^{-1}. □

Theorem 8.8.4. *For any super-exponential function f and any finite alphabet Σ, there exists a positive integer n such that there are no $f(l)$-distance ε-synchronization strings of length n or more over Σ.*

Proof. Consider a substring of length l in a given string over alphabet Σ. There are $|\Sigma|^l$ many possible assignments for such substring. Since f is a super-exponential function, for sufficiently large $l \geq \varepsilon^{-1}$, $\frac{f(l)}{l} \geq |\Sigma|^l$. For such l, consider a string of length $n \geq f(l)$. Split the first $f(l)$ elements into $\frac{f(l)}{l}$ blocks of length l. As $\frac{f(l)}{l} > |\Sigma|^l$, two of those blocks have to be identical. As l was assumed to be larger than ε^{-1}, this violates $f(l)$-distance ε-synchronization property for those two blocks and therefore finishes the proof. □

8.9 Infinite long-Distance Synchronization Strings: Efficient Constructions

In this section, we introduce and discuss the construction of infinite long-distance synchronization strings. The definition of c-long-distance ε-synchronization property strongly depends on the length of the string. This definition requires any two neighboring intervals as

well as any two intervals of aggregated length of $c \log n$ or more to satisfy ε-synchronization property. A natural generalization of this property to infinite strings would be to require similar guarantee to hold over all prefixes of it.

Definition 8.9.1 (Infinite Long-Distance Synchronization Strings)**.** *An infinite string S is called a* c-long-distance ε-synchronization string *if any prefix of S like $S[1, n]$ is a c-long-distance ε-synchronization string of length n.*

We prove infinite long distance synchronization strings exist and provide efficient constructions for them. We prove this by providing a structure similar to the one proposed in Theorem 8.3.16 that constructed an infinite ε-synchronization string using finite ε-synchronization strings.

Lemma 8.9.2. *Let $\mathcal{A}(n)$ be an algorithm that computes a c-long-distance ε-synchronization string $S \in \Sigma^n$ in $T(n)$ time. Further, let $\mathcal{A}_p(n, i)$ be an algorithm that computes ith position of a c-long-distance ε-synchronization string of length n in $T_p(n)$. Then, for any integer number $m \geq 2$, one can compose algorithms $\mathcal{A}'(n)$ and $\mathcal{A}'_p(i)$ that compute $S'[1, n]$ and $S'[i]$ respectively where S' is an infinite c-long-distance $\left(\varepsilon + \frac{4}{c \log m}\right)$-synchronization string over $\Sigma \times \Sigma$. Further, $\mathcal{A}'(n)$ and $\mathcal{A}'_p(i)$ run in $\min\{T(m^n), n \cdot T_p(m^n)\}$ and $T_p(m^i)$ time respectively.*

Proof. We closely follow the steps we took in Theorem 8.3.16, except, instead of using geometrically increasing synchronization strings in construction of U and V, we will use c-long-distance ε-synchronization strings whose length increase in the form of a tower function. We define the tower function $\text{tower}(p, i)$ for $p \in \mathbb{R}, i \in \mathbb{Z}^+$ recursively as follows: Let $\text{tower}(p, 1) = p$ and for $i > 1$, $\text{tower}(p, i) = p^{\text{tower}(p,i-1)}$. Then, we define two infinite strings U and V as follows:

$$U = (S_m, S_{m^{m^m}}, \dots), \qquad V = (S_{m^m}, S_{m^{m^{m^m}}}, \dots).$$

where S_l is a c-long-distance ε-synchronization string of length l. We define the infinite string T as the point by point concatenation of U and V.

We now show that this string satisfies the c-long-distance $\left(\varepsilon + \frac{4}{c \log m}\right)$-synchronization property. We define *turning points* $\{q_i\}_{i=1}^{\infty}$ in the same manner as we did in Theorem 8.3.16, i.e., the indices of T where a $S_{\text{tower}(m,i)}$ starts. Let q_i be the index where $S_{\text{tower}(m,i+1)}$ starts.

Consider two intervals $[i_1, j_1)$ and $[i_2, j_2)$ where $j_1 \leq i_2$ and $(j_1 - i_1) + (j_2 - i_2) \geq c \log j_2$. Let k be an integer for which $q_k < j_2 \leq q_{k+1}$. Then, $(j_1 - i_1) + (j_2 - i_2) \geq c \log j_2 \geq c \log(\text{tower}(m, k)) = c \log m \cdot \text{tower}(m, k-1)$. Note that all but $\text{tower}(m, k-1) + \text{tower}(m, k-3) + \cdots \leq 2 \cdot \text{tower}(m, k-1)$ elements of $T[i_1, j_1) \cup T[i_2, j_2)$ lie in

$T[q_{k-1}, q_{k+1})$ which is covered by $S_{\text{tower}(m,k-1)}$. Therefore, for $l = (j_1 - i_1) + (j_2 - i_2)$

$$\begin{aligned}
\text{ED}(T[i_1, j_1), T[i_2, j_2)) &\geq \text{ED}(T[\max\{i_1, q_{k-1}\}, j_1), T[i_2, j_2)) - 2 \cdot \text{tower}(m, k-1) \\
&\geq (1-\varepsilon) \cdot [(j_2 - i_2) + (j_1 - \max\{i_1, q_{k-1}\})] - 2 \cdot \text{tower}(m, k-1) \\
&\geq (1-\varepsilon) \cdot [l - 2 \cdot \text{tower}(m, k-1)] - 2 \cdot \text{tower}(m, k-1) \\
&\geq (1-\varepsilon) \cdot l - 4 \cdot \text{tower}(m, k-1) \\
&\geq \left(1 - \varepsilon - \frac{4 \cdot \text{tower}(m, k-1)}{l}\right) \cdot l \\
&\geq \left(1 - \varepsilon - \frac{4}{c \log m}\right) \cdot l
\end{aligned}$$

Further, any two neighboring intervals $[i_1, i_2)$ and $[i_2, i_3)$ where $i_3 - i_1 < c \log i_3$ and $k \leq i_3 < k+1$, $[i_1, i_3)$ completely lies in S_{k-1} and therefore ε-synchronization property for short neighboring intervals holds as well. Thus, this string satisfies infinite c-long-distance $\left(\varepsilon + \frac{4}{c \log m}\right)$-synchronization property.

Finally, to compute index i of infinite string T constructed as mentioned above, one needs to compute a single index of two finite c-long-distance ε-synchronization strings of length m^i or less. Therefore, computing $T[i]$ takes $T_p(m^i)$. This also implies that $T[1, n]$ can be computed in $n \cdot T_p(m^n)$. Clearly, on can also compute $T[1, n]$ by computing all finite strings that appear within the first n elements. Hence, $T[1, n]$ is computable in $\min\{T(m^n), n \cdot T_p(m^n)\}$. □

Utilizing the construction proposed in Lemma 8.9.2 with $m = 2$ along with the highly-explicit finite $O_\varepsilon(1)$-long-distance $\frac{\varepsilon}{2}$-synchronization string construction introduced in Theorem 8.3.15, results in the following infinite string construction:

Theorem 8.9.3. *For any constant $0 < \varepsilon < 1$ there is a deterministic algorithm which computes ith position of an infinite c-long-distance ε-synchronization string S over an alphabet of size $|\Sigma| = \varepsilon^{-O(1)}$ where $c = O_\varepsilon(1)$ in $O_\varepsilon(i)$ time. This implies a quadratic time construction for any prefix of such string.*

Chapter 9

Appoximating Edit Distance via Indexing: Near-Linear Time Codes

In Chapter 9, we introduce *fast-decodable indexing schemes for edit distance* which can be used to speed up edit distance computations to near-linear time if one of the strings is indexed by an indexing string I. In particular, for every length n and every $\varepsilon > 0$, one can, in near-linear time, construct a string $I \in \Sigma'^n$ with $|\Sigma'| = O_\varepsilon(1)$, such that, indexing any string $S \in \Sigma^n$ with I (i.e., concatenating S symbol-by-symbol with I) results in a string $S' \in \Sigma''^n$ where $\Sigma'' = \Sigma \times \Sigma'$ for which edit distance computations are easy, i.e., one can compute a $(1+\varepsilon)$-approximation of the edit distance between S' and any other string in $O(n\mathsf{polylog}(n))$ time.

Our indexing schemes can be used to improve the decoding complexity of the state-of-the-art error correcting codes for insertions and deletions. In particular, they lead to near-linear time decoding algorithms for the insertion-deletion codes from Chapter 3 and faster decoding algorithms for list-decodable insertion-deletion codes from Chapter 4. Interestingly, the latter codes are a crucial ingredient in the construction of fast-decodable indexing schemes.

9.1 Introduction

9.1.1 (Near) Linear-Time Codes

The seminal works of Shannon, Hamming, and others in the late 40s and early 50s established a good understanding of the optimal rate/distance tradeoffs achievable existentially and over the next decades, near-MDS codes achieving at least polynomial time decoding and encoding procedures were put forward. Since then, lowering the computational complexity has been an important goal of coding theory. Particularly, the 90s saw a big push, spearheaded by Spielman, to achieve (near) linear coding complexities: in a breakthrough in 1994, Sipser and Spielman [SS96] introduced expander codes and derived linear codes with some constant distance and rate that are decodable (but not encodable) in linear time. In 1996 Spielman [Spi96] build upon these codes to derive asymptotically good error correcting codes that are encodable and decodable in linear time. As for codes with better rate distance trade-off, Alon et al. [AEL95] obtained near-MDS error correcting codes that were decodable from erasures in linear time. Finally, in 2004, Guruswami and Indyk [GI05] provided near-MDS error correcting codes for any rate than can be decoded in linear time from any combination of symbol erasures and symbol substitutions.

9.1.2 Codes for Insertions and Deletions

Similar questions on communication and computational efficiencies hold for synchronization codes, i.e., codes that correct from symbol insertions and symbol deletions. As a matter of fact, an analogous flow of progress can be recognized for synchronization codes. The study of synchronization codes started with the work of Levenshtein [Lev65] in the 60s. In 1999, Schulman and Zuckerman [SZ99] gave the first (efficient) synchronization code with constant distance and rate. Only recently, synchronization codes with stronger communication efficiency have been found. Guruswami et al. [GW17, GL16] introduced the first synchronization codes in the asymptotically small or large noise regimes by giving efficient codes which achieve a constant rate for noise rates going to one and codes which provide a rate going to one for an asymptotically small noise rate. The work presented in Chapters 3 and 4 of this thesis, was able to finally achieve efficient synchronization codes with the optimal (near-MDS) rate/distance tradeoff, for any rate and distance using synchronization strings for unique-decoding and list-decoding settings.

All of the codes mentioned so far have decoders with large polynomial complexity between $\Omega(n^2)$ and $O(n^{O(1/\varepsilon)})$. The only known insertion-deletion codes prior to this work with subquadratic time decoders are given in Chapter 8. Unfortunately, these codes only work for $\delta \in \left(0, \frac{1}{3}\right)$ fraction of errors while achieving a rate of $1 - 3\delta - \varepsilon$ (instead of the desired $1 - \delta - \varepsilon$).

In this work, we take the natural next step and address the problem of finding near-linear time encodable/decodable (near-MDS) codes for insertions and deletions.

9.1.3 Quadratic Edit Distance Computation Barrier

Many of the techniques developed for constructing efficient regular error correcting codes also apply to synchronization strings. Indeed, the synchronization string based constructions show that this can largely be done in a black-box manner. However, there is a serious new barrier that arises in the setting of synchronization errors if one tries to push computational complexities below n^2. This barrier becomes apparent once one notices that decoding an error correcting code is essentially doing a distance minimization where the appropriate distance in the synchronization setting is the edit distance[1]. As we discuss below, merely computing the edit distance between two strings (the input and a candidate output of the decoder) in subquadratic time is a well known hard problem. An added dimension of challenge in our setting is that we must first select the candidate decoder outputs among exponentially many codewords.

A simple algorithm for computing the edit distance of two given strings is the classic Wagner-Fischer dynamic programming algorithm that runs in quadratic time. Improving the running time of this simple algorithm has been a central open problem in computer science for decades (e.g. [CKK72]). Yet to date, only a slightly faster algorithm ($O(n^2/\log^2 n)$) due to Masek and Paterson [MP80] is known. Furthermore, a sequence of complexity breakthroughs from recent years suggests that a near-quadratic running time may in fact be optimal [AWW14, BI18, ABW15, BK15, AHWW16] (under the Strong Exponential Time Hypothesis (SETH) or related assumptions). In order to obtain subquadratic running times, computer scientists have considered two directions: moving beyond worst-case instances, and allowing approximations.

Beyond worst case

Edit distance computation is known to be easier in several special cases. For the case where edit distance is known to be at most k, Ukkonen [Ukk85] provided an $O(nk)$ time algorithm and Landau et al. [LMS98] improved upon that with an $O(n+k^2)$ time algorithm. For the case where the longest common subsequence (LCS) is known to be at most L, Hirschberg [Hir77] gave an algorithm running in time $O(n \log n + Ln)$. Following a long line of works, Gawrychowski [Gaw12] currently has the fastest algorithm for the special case of strings that can be compressed as small *straight-line programs* (SLP). Andoni and Krauthgamer [AK12] obtain efficient approximations to edit distance for the case where the strings are perturbed a-la smoothed analysis. Goldwasser and Holden [GH17b] obtain subquadratic algorithms when the input is augmented with auxiliary correlated strings.

Other special cases have also been considered (see also [BK18]), but the work closest to ours is by Hunt and Szymanski [HS77], who obtained a running time of $O((n+r)\log n)$ for the special case where there are r "matching pairs", i.e. pairs of identical characters (see also Section 9.3.2). While we directly build on their algorithm, note that there is an obstacle to applying it in our setting: for a constant size alphabet, we expect that a constant fraction of all n^2 pairs of characters will be matching, i.e. $r = \Theta(n^2)$.

[1]We define the *edit distance* between two strings S, S' as the minimum number of character insertions and deletions required to transform S to S'. Note that this is slightly different (but closely related) to the more standard definition which also allows character substitutions.

Approximation algorithms

There is a long line of works on efficient approximation algorithms for edit distance in the worst case [BYJKK04, BES06, AKO10, AO12, BEG+18, CDG+18]. First, it is important to note that even after the recent breakthrough of Chakraborty et al. [CDG+18], it is not known how to obtain approximation factors better than 3 (see also discussion in [Rub18]). Furthermore, our running time is much faster than Chakraborty et al.'s [CDG+18] and even faster than the near-linear time approximations of [AKO10, AO12]. The best known approximation factor in time $O(n\mathsf{polylog}(n))$ is still worse than $n^{1/3}$ [BES06].

In terms of techniques, our algorithm is most closely inspired by the window-compatible matching paradigm introduced by the recent quantum approximation algorithm of Boroujeni et al. [BEG+18] (a similar idea was also used by Chakraborty et al. [CDG+18]).

A new ray of hope

In this work we combine both approaches: namely we allow for (arbitrarily good) approximation, and also restrict our attention to the special case of computing the edit distance between a worst case input and a codeword from our code. The interesting question thus becomes if there is a way to build enough structure into a string (or a set of strings/codewords) that allows for fast edit distance computations. Given the importance and pervasiveness of edit distance problems we find this to be a question of interest way beyond its applicability to synchronization codes. An independent work of Kuszmaul [Kus19] also employs the combination of the two approaches and provides a near-linear time algorithm for approximating the edit distance between a pseudo-random string and an arbitrary one within a constant factor.

9.2 Our Results

In this chapter, we introduce a simple and generic structure that achieves this goal. In particular, we will show that there exist strings over a finite alphabet that, if one indexes any given string S with them, the edit distance of the resulting string to any other string S' can be approximated within a $1+\varepsilon$ factor in near-linear time. This also leads to breaking the quadratic decoding time barrier for insertion-deletion codes with near-optimal communication efficiency.

We start with a formal definition of *string indexing* followed by the definition of an *indexing scheme*.

Definition 9.2.1 (String Indexing or Coordinate-Wise String Concatenation)**.** *Let $S \in \Sigma^n$ and $S' \in \Sigma'^n$ be two strings of length n over alphabets Σ and Σ'. The* coordinate-wise concatenation of S and S' *or* S indexed by S' *is a string of length n over alphabet $\Sigma \times \Sigma'$ whose ith element is (S_i, S'_i). We denote this string with $S \times S'$.*

Definition 9.2.2 (Indexing Scheme)**.** *The pair $(I, \widetilde{\mathrm{ED}}_I)$ consisting of string $I \in \Sigma_{\mathrm{Index}}^n$ and algorithm $\widetilde{\mathrm{ED}}_I$ is an* ε-indexing scheme *if for any string $S \in \Sigma^n$ and $S' \in [\Sigma \times \Sigma_{\mathrm{Index}}]^n$, $\widetilde{\mathrm{ED}}_I(S \times I, S')$ outputs a set of up to $(1+\varepsilon)\mathrm{ED}(S \times I, S')$ symbol insertions and symbol*

deletions over $S \times I$ that turns it into S'. The $\mathrm{ED}(\cdot)$ *notation represents the edit distance function.*

The main result of this work is on the existence of indexing schemes that facilitate approximating the edit distance in near-linear time.

Theorem 9.2.3. *For any $\varepsilon \in (0,1)$ and integer n, there exist a string $I \in \Sigma_{\text{Index}}^n$ and an algorithm $\widetilde{\text{ED}}_I$ where $(I, \widetilde{\text{ED}}_I)$ form an ε-indexing scheme, $|\Sigma_{\text{Index}}| = \exp\left(\frac{\log(1/\varepsilon)}{\varepsilon^3}\right)$, $\widetilde{\text{ED}}_I$ runs in $O_\varepsilon(n\mathsf{polylog}(n))$ time, and I can be constructed in $O_\varepsilon(n\mathsf{polylog}(n))$ time.*

9.2.1 Applications

One application of indexing schemes that we introduce in this work is in enhancing the design of insertion-deletion codes (insdel codes) from Chapters 3 and 4. The construction of codes from Chapters 3 and 4 consist of indexing each codeword of some appropriately chosen error correcting code with symbols of a synchronization string which, in the decoding procedure, will be used to recover the position of received symbols. As we will recapitulate in Section 9.7, this procedure of recovering the positions consists of several longest common subsequence computations between the utilized synchronization string and some other version of it that is altered by a number of insertions and deletions. This fundamental step resulted in an $\Omega(n^2)$ decoding time complexity for codes in Chapters 3 and 4.

Using the ε-indexing schemes in this chapter, we will modify constructions of Chapters 3 and 4 so that the above-mentioned longest common subsequence computations can be replaced with approximations of the longest common subsequence (using Theorem 9.2.3) that run in near-linear time. The following theorem, that improves Theorem 3.1.1 with respect to the decoding complexity, gives an insertion-deletion code for the entire range of distance that approaches the Singleton bound and is decodable in near-linear time.

Theorem 9.2.4. *For any $\varepsilon > 0$ and $\delta \in (0,1)$ there exists an encoding map $E : \Sigma^k \to \Sigma^n$ and a decoding map $D : \Sigma^* \to \Sigma^k$, such that, if $\mathrm{ED}(E(m), x) \leq \delta n$ then $D(x) = m$. Further, $\frac{k}{n} > 1 - \delta - \varepsilon$, $|\Sigma| = \exp\left(\varepsilon^{-4}\log(1/\varepsilon)\right)$, and E and D are explicit and can be computed in linear and near-linear time in terms of n respectively.*

A very similar improvement is also applicable to the design of list-decodable insertion-deletion codes from Chapter 4 as they also utilize indexed synchronization strings and a similar position recovery procedure along with an appropriately chosen list-recoverable code. (See Definition 9.3.2) In this case, we obtain list-decodable insertion-deletion codes that match the fastest known list-recoverable codes in terms of decoding time complexity.

Theorem 9.2.5. *For every $0 < \delta, \varepsilon < 1$ and $\varepsilon_0, \gamma > 0$, there exists a family of list-decodable codes that can protect against δ-fraction of deletions and γ-fraction of insertions and achieves a rate of at least $1 - \delta - \varepsilon$ over an alphabet of size $O_{\varepsilon_0,\varepsilon,\gamma}(1)$. There exists a randomized decoder for these codes with list size $L_{\varepsilon_0,\varepsilon,\gamma}(n) = \exp\left(\exp\left(\exp\left(\log^* n\right)\right)\right)$, $O(n^{1+\varepsilon_0})$ encoding and decoding complexities, and decoding success probability 2/3.*

Both Theorems 9.2.4 and 9.2.5 are built upon the fact that if one indexes a synchronization string with an appropriate indexing scheme, the resulting string will be a synchronization string that is decodable in near-linear time.

9.2.2 Other Results, Connection to List-Recovery, and the Organization

In the rest of this chapter, we first provide some preliminaries and useful lemmas from previous works in Section 9.3. In Section 9.4, we introduce the construction of our indexing schemes and prove Theorem 9.2.3. The construction of these indexing schemes utilize insertion-deletion codes that are list-decodable from large fractions of deletions and insertions. We use list-decodable codes from Chapter 4 for that purpose, which themselves use list-recoverable codes as a core building block. Therefore, the quality of indexing schemes that we provide, namely, time complexity and alphabet size, greatly depend on utilized list-recoverable codes and can be improved following the prospective advancement of list-recoverable codes in the future.

In Section 9.5, we enhance the structure of the indexing scheme from Theorem 9.2.3 and provide Theorem 9.5.1 that describes a construction of indexing schemes using $(\varepsilon, \frac{1}{\varepsilon}, L)$-list-recoverable codes as a black-box. This result opens the door to potentially reduce the polylogarithmic terms in the time complexity of indexing schemes from Theorem 9.2.3 by future developments in the design of list-recoverable codes. For instance, finding near-linear time $(\varepsilon, \frac{1}{\varepsilon}, \mathsf{polylog}(n))$-list recoverable codes leads to indexing schemes that run in $O(n\mathsf{poly}(\log\log n))$ time via Theorem 9.5.1.

As of the time of writing this thesis, no such list-recoverable code is known. However, a recent work of Hemenway, Ron-Zewi, and Wootters [HRZW19] presents list-recoverable codes with $O\left(n^{1+\varepsilon_0}\right)$ time probabilistic decoders for any $\varepsilon_0 > 0$ that are appropriate for the purpose of being utilized in the construction of indexing schemes as outlined in Theorem 9.5.1. In Section 9.6, we use such codes with the indexing scheme construction method of Theorem 9.5.1 to provide a randomized indexing scheme with $O(n\log^{\varepsilon_0} n)$ time complexity for any chosen $\varepsilon_0 > 0$.

Then, in Section 9.7, we discuss the application of indexing schemes in the design of insertion-deletion codes. We start by Theorem 9.7.1 that enhances synchronization strings by using them along with indexing schemes and, therefore, enables us to reduce the time complexity of the position recovery subroutine of the decoders of codes from Chapters 3 and 4 to near-linear time. In Section 9.7.2, we discuss our results for uniquely-decodable codes and prove Theorem 9.2.4. At the end, in Section 9.7.3, we address construction of list-decodable synchronization codes using indexing schemes. We start by Theorem 9.7.4 that gives a black-box conversion of a given list-recoverable code to a list-decodable insertion-deletion code by adding only a near-linear time overhead to the decoding complexity and, therefore, paves the path to obtaining insertion-deletion codes that are list-decodable in near-linear time upon the design of near-linear time list-recoverable codes. We use this conversion along with list-recoverable codes of [HRZW19] to prove Theorem 9.2.5.

The notion of edit distance that we study in this thesis is defined as the smallest number of insertions and deletions needed to convert one string to another. In Section 9.8, we extend our techniques to approximate more general edit-distance-like metrics, most notably, the Levenshtein distance which allows symbol substitutions as well as symbol insertions and deletions as a unit of modification.

9.3 Preliminaries and Notation

In this section, we provide definitions and preliminaries that will be useful throughout the rest of this chapter.

9.3.1 Synchronization Strings

We start by some essential definitions and lemmas regarding synchronization strings. In Chapter 3, we showed that ε-synchronization matchings satisfy the following property.

Theorem 9.3.1 (Theorem 3.5.2)**.** *Let S be an ε-synchronization string of length n and $1 \leq i_1 < i_2 < \cdots < i_l \leq n$ and $1 \leq j_1 < j_2 < \cdots < j_l \leq n$ be integers so that $S(i_k) = S(j_k)$ but $i_k \neq j_k$ for all $1 \leq k \leq l$. Then $l \leq \varepsilon n$.*

In Chapter 4, we proposed a construction of list-decodable insertion-deletion codes by indexing the codewords of a list-recoverable code with symbols of a synchronization string. As we will use similar techniques and ideas throughout this chapter, we formally define list-recoverable codes and review the main result of Chapter 4 in the following.

Definition 9.3.2 (List-recoverable codes)**.** *Code $\mathcal{C}$ with encoding function $\mathrm{Enc} : \Sigma^{nr} \to \Sigma^n$ is called (α, l, L)-list recoverable if for any collection of n sets $S_1, S_2, \cdots, S_n \subset \Sigma$ of size l or less, there are at most L codewords for which more than αn elements appear in the list that corresponds to their position, i.e.,*

$$|\{x \in \mathcal{C} \mid |\{i \in [n] \mid x_i \in S_i\}| \geq \alpha n\}| \leq L.$$

Theorem 9.3.3 (Restatement of Theorem 4.1.1)**.** *For every $0 < \delta, \varepsilon < 1$ and $\gamma > 0$, there exist a family of list-decodable insdel codes that can protect against δ-fraction of deletions and γ-fraction of insertions and achieves a rate of $1 - \delta - \varepsilon$ or more over an alphabet of size $\left(\frac{\gamma+1}{\varepsilon^2}\right)^{O\left(\frac{\gamma+1}{\varepsilon^3}\right)} = O_{\gamma,\varepsilon}(1)$. These codes are list-decodable with lists of size $L_{\varepsilon,\gamma}(n) = \exp\left(\exp\left(\exp\left(\log^* n\right)\right)\right)$, and have polynomial time encoding and decoding complexities.*

By choosing $\delta = \gamma = 1 - \epsilon$ and $\varepsilon = \epsilon/2$ in Theorem 4.1.1, we derive the following corollary.

Corollary 9.3.4. *For any $0 < \varepsilon < 1$, there exists an alphabet Σ_ε with size $\exp(\varepsilon^{-3} \log 1/\varepsilon)$ and an infinite family of insertion-deletion codes, $\mathcal{C}$, that achieves a rate of $\frac{\varepsilon}{2}$ and is L-list-decodable from any $(1-\varepsilon)n$ deletions and $(1-\varepsilon)n$ insertions in polynomial time where $L = \exp(\exp(\exp(\log^* n)))$.*

9.3.2 Non-crossing Matchings

The last element that we utilize as a preliminary tool in this chapter is an algorithm provided in a work of Hunt and Szymanski [HS77] to compute the maximum *non-crossing matching* in a bipartite graph. Let G be a bipartite graph with an ordering for vertices in each part. A non-crossing matching in G is a matching in which edges do not intersect.

Definition 9.3.5 (Non-Crossing Matching)**.** *Let G be a bipartite graph with ordered vertices $u_1, u_2, \cdots, u_m$ and $v_1, v_2, \cdots, v_n$ in each part. A non-crossing matching is a subset of edges of G like*

$$\{(u_{i_1}, v_{j_1}), (u_{i_2}, v_{j_2}), \cdots, (u_{i_l}, v_{j_l})\}$$

where $i_1 < i_2 < \cdots < i_l$ and $j_1 < j_2 < \cdots < j_l$.

In this chapter, we use an algorithm by Hunt and Szymanski [HS77] that essentially computes the largest non-crossing matching in a given bipartite graph.

Theorem 9.3.6 (Theorem 2 of Hunt and Szymanski [HS77])**.** *Let G be a bipartite graph with n ordered vertices in each part and r edges. There is an algorithm that computes the largest non-crossing matching of G in $O((n+r)\log\log n)$.*

9.4 Near-Linear Edit Distance Computations via Indexing

We start by a description of the string that will be used in our indexing scheme. Let $\mathcal{C}$ be an insertion-deletion code over alphabet $\Sigma_{\mathcal{C}}$, with block length N, and rate r that is L-list decodable from any $N(1-\varepsilon)$ deletions and $N(1-\varepsilon)$ insertions in $T_{\mathrm{Dec}_{\mathcal{C}}}(N)$ for some sufficiently small $\varepsilon > 0$. We construct the indexing sequence I by simply concatenating the codewords of $\mathcal{C}$. The construction of such indexing sequence resembles long-distance synchronization strings from Chapter 4.

Throughout this section, we consider string S of length $N \cdot |\Sigma_{\mathcal{C}}|^{Nr}$ that consists of coordinate-wise concatenation of a content string m and the indexing string I. In other words, $S_i = (m_i, I_i)$. We will provide algorithms that approximate the edit distance of S to a given string S'.

Consider the longest common subsequence between S and S'. One can represent such common subsequence by a matching $\mathcal{M}_{LCS}$ with non-crossing edges in a bipartite graph with two parts of size $|S|$ and $|S'|$ where each vertex corresponds to a symbol in S or S' and each edge corresponds to a pair of identical symbols in the longest common subsequence.

Note that one can turn S into S' by simply deleting any symbol that corresponds to an unmatched vertex in S and then inserting symbols that correspond to the unmatched vertices in S'. Therefore, the edit distance between S and S' is equal to the number of non-connected vertices in that graph. To provide a $(1+\varepsilon)$ edit distance approximation as described in Theorem 9.2.3, one only needs to compute a common subsequence, or equivalently, a non-crossing matching between S and S' in which the number of unmatched vertices does not exceed a $1+\varepsilon$ multiplicative factor of $\mathcal{M}_{LCS}$'s.

We start by an informal intuitive justification of the algorithm. The algorithm starts by splitting the string S' into blocks of length N in the same spirit as S. We denote ith such block by $S'(i)$ and the ith block of S by $S(i)$. Note that the blocks of S are codewords of an insertion-deletion code with high distance indexed by m ($S(i) = \mathcal{C}(i) \times m[N(i-1), Ni-1]$). Therefore, one might expect that any block of S that is not significantly altered by insertions and deletions, (1) appears in a set of consecutive blocks in S' and (2) has a small edit distance to at least one of those blocks.

Following this intuition, our proposed algorithm works thusly: For any block of S' like $S'(i)$, the algorithm uses the list decoder of $\mathcal{C}$ to find all (up to L) blocks of S that can be turned into $S'(i)$ by $N(1-\varepsilon)$ deletions and $N(1-\varepsilon)$ insertions ignoring the content portion on S'. In other words, let $S'(i) = C'_i \times m'[N(i-1), Ni-1]$. We denote the set of such blocks by $\mathrm{Dec}_{\mathcal{C}}(C'_i)$. Then, the algorithm constructs a bipartite graph G with $|S|$ and $|S'|$ vertices on each side (representing symbols of S and S') as follows: a symbol in $S'(i)$ is connected to all identical symbols in the blocks that appear in $\mathrm{Dec}_{\mathcal{C}}(C'_i)$ or any block that is in their $w = O\left(\frac{1}{\varepsilon}\right)$ neighborhood, i.e., is up to $O\left(\frac{1}{\varepsilon}\right)$ blocks away from at least one of the members of $\mathrm{Dec}_{\mathcal{C}}(C'_i)$.

Note that any non-crossing matching in G corresponds to some common subsequence between S and S' because G's edges only connect identical symbols. In the next step, the algorithm finds the largest non-crossing matching in G, $\mathcal{M}_{ALG}$, and outputs the corresponding set of insertions and deletions as the output. We will use the algorithm proposed by Hunt and Szymanski [HS77] (see Theorem 9.3.6) to find the largest non-crossing matching. A formal description of the algorithm is available in Algorithm 11.

Algorithm 11 $(1+11\varepsilon)$-Approximation for Edit Distance

```
1: procedure ED-Approx(S, S', N, Dec_C(·))
2:     Make empty bipartite graph G with parts of size (|S|, |S'|)
3:     w = 1/ε
4:     for each S'(i) = C'_i × m'[N(i−1), Ni−1] do
5:         List ← Dec_C(C'_i)
6:         for each j ∈ List do
7:             for k ∈ [j − w, j + w] do
8:                 Connect pairs of vertices in G that correspond to identical symbols in
       S(k) and S'(i).
9:     M_ALG ← Largest non-crossing matching in G (Using Theorem 9.3.6)
10:    return M_ALG
```

9.4.1 Analysis

We now proceed to the analysis of approximation guarantee and time complexity of Algorithm 11.

Theorem 9.4.1. *For $n = \max(|S|, |S'|)$, the running time of Algorithm 11 is $O\left(\frac{n}{N} \cdot T_{\mathrm{Dec}_{\mathcal{C}}}(N) + \frac{NL}{\varepsilon} \cdot n \log\log n\right)$.*

Proof. The algorithm starts by using the decoder for any block in S' which takes a total of $\frac{n}{N} \cdot T_{\mathrm{Dec}_{\mathcal{C}}}(N)$ time. Further, construction of G will take $O\left(nLN\frac{1}{\varepsilon}\right)$. G has no more than $n \cdot NL \cdot w = O\left(\frac{nNL}{\varepsilon}\right)$ edges. Thus, by using Hunt and Szymanski's [HS77] algorithm (Theorem 9.3.6), the maximum non-crossing matching in G can be computed in $O\left((n + \frac{nNL}{\varepsilon}) \log\log n\right) = O\left(\frac{NL}{\varepsilon} \cdot n \log\log n\right)$. □

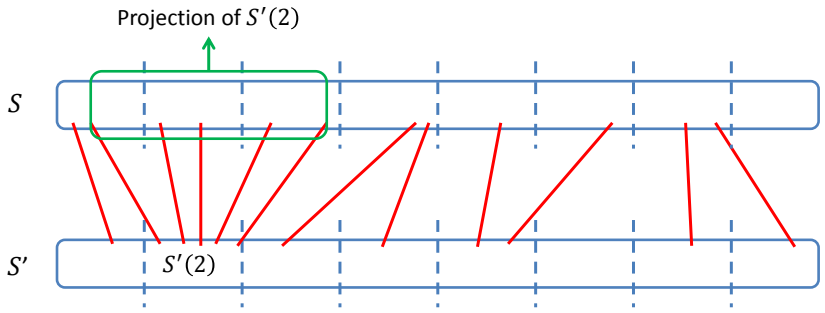


Figure 9.1: An example of a matching between S and S' depicting the projection of $S'(2)$. This matching is 3-window-limited.

Before providing the analysis for the approximation ratio of Algorithm 11, we define the following useful notions.

Definition 9.4.2 (Projection)**.** *Let $\mathcal{M}$ be a non-crossing matching between S and S'. The* projection of $S'(i)$ under $\mathcal{M}$ *is defined to be the substring of S between the leftmost and the rightmost element of S that are connected to $S'(i)$ in $\mathcal{M}$. (see Fig. 9.1 for an example)*

Definition 9.4.3 (Window Limited)**.** *A non-crossing matching between S and S' is called* w-window-limited *if the projection of any block of S' fits in w consecutive blocks of S.*

The definition of window-limited matchings is inspired by the window-compatibility notion from [BEG+18].

Theorem 9.4.4. *For $0 < \varepsilon < \frac{1}{21}$, Algorithm 11 computes a set of up to $(1+11\varepsilon)\cdot \mathrm{ED}(S, S')$ insertions and deletions that turn S into S'.*

Proof. Let ED_{ALG} denote the edit distance solution obtained by the matching suggested by Algorithm 11. We will prove that $\mathrm{ED}_{ALG} \leq (1 + 11\varepsilon) \cdot \mathrm{ED}(S, S')$ in the following two steps:

1. Let $\mathcal{M}_W$ be the largest $w = \left(\frac{1}{\varepsilon} + 1\right)$-window-limited matching between S and S' and ED_W be its count of unmatched vertices. In the first step, we show the following.

$$\mathrm{ED}_W \leq (1 + 3\varepsilon)\mathrm{ED}(S, S') \tag{9.1}$$

   To prove this, consider $\mathcal{M}_{LCS}$, the matching that corresponds to the longest common subsequence. Then, we modify this matching by deleting all the edges connected to any block $S'(i)$ that violates the w-window-limited requirement. In other words, if the projection of $S'(i)$ spans over at least $w+1$ blocks in S, we remove all the edges with one endpoint in $S'(i)$. Note that removing the edges connected to $S'(i)$ might increase the number of unmatched vertices in the matching by $2N$. However, as projection of $S'(i)$ spans over at least $w+1$ blocks in S, one can assign all the originally unmatched vertices in that projection, which are at least $(w-1)\cdot N - N \geq (w-2)N$, to the newly introduced unmatched edges as an "approximation budget". Note that this

assignment is mutually exclusive since projections of two distinct blocks of S' are disjoint. Therefore, the above-mentioned removal procedure increases the number of unmatched vertices by a multiplicative factor no larger than $\frac{(w-2)N+2N}{(w-2)N} = \frac{w}{w-2} = \frac{1+\varepsilon}{1-\varepsilon} \leq 1+3\varepsilon$ for $\varepsilon \leq \frac{1}{3}$.

Note that the matching obtained by the above-mentioned removal procedure is a w-window-limited matching and, therefore, has at least ED_W unmatched vertices by the definition of $\mathcal{M}_W$. Hence, Eq. (9.1) is proved.

2. In the second step, we show that

$$\mathrm{ED}_{ALG} \leq (1+7\varepsilon)\mathrm{ED}_W. \tag{9.2}$$

Similar to Step 1, consider the largest w-window-limited matching $\mathcal{M}_W$ and then modify it by removing all the edges connected to any block $S'(i)$ that has less than εN edges to any block in S. Again, we prove an approximation ratio by exclusively assigning some of the unmatched vertices in $\mathcal{M}_W$ to each $S'(i)$ that we choose to remove its edges.

Consider some $S'(i)$ that has less than εN edges to any block in S. We assign all unmatched vertices in $S'(i)$ and all unmatched vertices in the projection of $S'(i)$ as the approximation budget for eliminated edges. Let B be the number of blocks in S that are contained or intersect with projection of $S'(i)$. As $S'(i)$ has less than εN edges to any block in S, the total number of removed edges is less than $NB\varepsilon$. This gives that there are at least $N-NB\varepsilon$ unmatched vertices within S' and $\max\{B-2,0\}\cdot N(1-\varepsilon)$ unmatched vertices in its projection that are assigned to $2NB\varepsilon$ new unmatched edges appearing as a result of removing $S'(i)$'s edges. Therefore, this process does not increase the number of unmatched vertices by a multiplicative factor more than $1+\frac{2NB\varepsilon}{(N-NB\varepsilon)+\max\{B-2,0\}\cdot N(1-\varepsilon)}$.

If $B=1$ or 2, the above approximation ratio can be bounded above by $1+\frac{2NB\varepsilon}{(N-NB\varepsilon)} \leq 1+\frac{4\varepsilon}{1-2\varepsilon} \leq 1+5\varepsilon$ for $\varepsilon \leq \frac{1}{10}$. Unless, $B \geq 3$, therefore the approximation ratio is less than $1+\frac{2NB\varepsilon}{(B-2)N(1-\varepsilon)} \leq 1+\frac{6\varepsilon}{1-\varepsilon} \leq 1+7\varepsilon$ for $\varepsilon \leq \frac{1}{7}$. Therefore, the edge removal process in Step 2 does not increase the number of unmatched vertices by a factor larger than $1+7\varepsilon$.

Note that the matching obtained after the above-mentioned procedure is a w-window limited one in which any block of S' that contains at least one edge, has more than $N\varepsilon$ edges to some block in S within its projection. Therefore, this matching is a subgraph of G. Since $\mathcal{M}_{ALG}$ is defined to be the largest non-crossing matching in G, the number of unmatched vertices in $\mathcal{M}_{ALG}$, ED_{ALG} is not larger than the ones in the matching we obtained in Step 2. Hence, proof of Eq. (9.2) is complete.

Combining Eqs. (9.1) and (9.2) implies the following approximation ratio.

$$\mathrm{ED}_{ALG} \leq (1+3\varepsilon)(1+7\varepsilon)\mathrm{ED}(S,S') \leq (1+11\varepsilon)\mathrm{ED}(S,S') \tag{9.3}$$

The last inequality holds for $\varepsilon \leq \frac{1}{21}$. □

9.4.2 Proof of Theorem 9.2.3

Proof. To prove this theorem, take $\varepsilon' = \frac{\varepsilon}{11}$. Further, take $\mathcal{C}$ as an insertion-deletion code from Theorem 4.1.1 with block length $N = c_0 \cdot \frac{\log n \cdot \varepsilon'^3}{(1-2\varepsilon')\log(1/\varepsilon')}$ and parameters $\delta_{\mathcal{C}} = \gamma_{\mathcal{C}} = 1-\varepsilon'$, $\varepsilon_{\mathcal{C}} = \varepsilon'$. (constant c_0 will be determined later)

According to Theorem 4.1.1, $\mathcal{C}$ is $O_\varepsilon(\exp(\exp(\exp(\log^* n))))$-list decodable from $(1-\varepsilon')N$ insertions and $(1-\varepsilon')N$ deletions, is over an alphabet of size $q_{\mathcal{C}} = \varepsilon'^{-O(1/\varepsilon'^3)} = \exp\left(\frac{\log(1/\varepsilon')}{\varepsilon'^3}\right)$, and has rate $r_{\mathcal{C}} = 1 - 2\varepsilon'$.

Construct string I according to the structure described in the beginning of Section 9.4 using $\mathcal{C}$ as the required list-decodable insertion-deletion code. Note that $|I| = N \cdot q_{\mathcal{C}}^{r_{\mathcal{C}} N} = N \cdot \exp\left(c_0 \cdot O(\log n)\right)$. Choosing an appropriate constant c_0 that cancels out the constants hidden in O-notation that originate from hidden constants in the alphabet size will lead to $|I| = Nn = O(n \log n)$. Truncate the extra elements to have string I of length n. As $\mathcal{C}$ is efficiently encodable, string I can be constructed in near-linear time.

Further, define algorithm $\widetilde{\mathrm{ED}}_I$ as follows. $\widetilde{\mathrm{ED}}_I$ takes $S \times I$ and S' and runs an instance of Algorithm 11 with $S \times I$, S', N, and the decoder of $\mathcal{C}$ as its input. Theorem 9.4.4 guarantees that $\widetilde{\mathrm{ED}}_I(S \times I, S')$ generates a set of at most $(1+11\varepsilon')\mathrm{ED}(S \times I, S') = (1+\varepsilon)\mathrm{ED}(S \times I, S')$ insertions and deletions over $S \times I$ that converts it to S'. Finally, Theorem 9.4.1 guarantees that $\widetilde{\mathrm{ED}}_I$ runs in

$$\begin{aligned} & O\left(\frac{n}{N} \cdot T_{\mathrm{Dec}_{\mathcal{C}}}(N) + \frac{NL}{\varepsilon} \cdot n \log\log n\right) \\ = \; & O_\varepsilon\left(\frac{n}{\log n} T_{\mathrm{Dec}_{\mathcal{C}}}(\log n) + n \log n \log\log n \exp(\exp(\exp(\log^* n)))\right) \\ = \; & O_\varepsilon(n\mathsf{poly}(\log n)) \end{aligned}$$

time. □

9.5 Enhanced Indexing Scheme

In Section 9.4, we provided an indexing scheme, using which, one can essentially approximate the edit distance by a $(1+\varepsilon)$ multiplicative factor for any $\varepsilon > 0$. Note that if code $\mathcal{C}$ that was used in that construction has some constant rate $r = O_\varepsilon(1)$, then $|S| = N \cdot |\Sigma_{\mathcal{C}}|^{Nr}$ and, therefore, $N = \Theta_\varepsilon\left(\frac{\log n}{r}\right)$. This makes the running time of Algorithm 11 from Theorem 9.4.1 $O\left(\frac{nr}{\log n} \cdot T_{\mathrm{Dec}_{\mathcal{C}}}(\log n) + \frac{\log n \cdot L}{\varepsilon} \cdot n \log\log n\right)$. As described in the proof of Theorem 9.2.3, using the efficient list-decodable codes from Corollary 9.3.4, one can obtain edit distance computations in $O_\varepsilon(n \cdot \mathrm{poly}(\log n) + n \log n \cdot \log\log n \cdot \exp\left(\exp(\exp(\log^* n))\right)) = O_\varepsilon(n \cdot \mathrm{poly}(\log n))$.

In this section, we try to enhance this running time by reducing the poly-logarithmic terms. To this end, we break down the factors in our construction and edit distance computation that contribute to the poly-logarithmic terms in the decoding time complexity.

1. **Edges in graph** G: The number of edges in graph G can be as high as $\Theta\left(\frac{nNL}{\varepsilon}\right) = \Theta(n \log n \cdot \mathrm{poly}(\log\log n))$ which, as discussed above, leads to an additive $n \log n \cdot$

$\log\log n \cdot \exp\left(\exp(\exp(\log^* n))\right)$ component. In Section 9.5.1, we will show that this component can be reduced to $O(n \cdot \text{poly}(\log\log n))$ by having two layers of indices via indexing each codeword of $\mathcal{C}$ with an indexing scheme as described in Section 9.4 (constructed based on some code of block length $O(\log\log n)$).

2. **Decoding complexity of code $\mathcal{C}$ from Corollary 9.3.4 ($T_{\text{Dec}_{\mathcal{C}}}(\cdot)$)**: As described in Section 9.3.1, list-decodable insdel codes from Theorem 4.1.1 are obtained by indexing codewords of a list-recoverable code with a synchronization string and their decoding procedure consist of (1) calculating a constant number of longest common subsequence computations, and (2) running the decoder of the list-recoverable code.

   Part (1) consumes quadratic time in terms of N. However, using the indexing schemes for approximating edit distance from Theorem 9.2.3, we will show in Theorem 9.7.4 that one can reduce the running time of part (1) to $O\left(\frac{n}{\log n} \cdot \log n \cdot \mathsf{poly}(\log\log n)\right) = O\left(n \cdot \mathsf{poly}(\log\log n)\right)$.

Applying the above-metioned enhancements to the structure of our indexing scheme will result in the black-box construction of indexing schemes using list-recoverable codes as formalized in the following theorem.

Theorem 9.5.1. *For any $\varepsilon \in (0,1)$, given a family of codes over alphabet Σ that are $\left(\frac{\varepsilon}{46}, \frac{276}{\varepsilon}, L(\cdot)\right)$-list recoverable in $T_{\text{Dec}}(\cdot)$ time and achieve a rate of $r = O_\varepsilon(1)$, one can construct an ε-indexing scheme $(I, \widetilde{\text{ED}}_I)$ with any positive length n over an alphabet of size $|\Sigma|^2 \times \exp\left(\frac{\log(1/\varepsilon)}{\varepsilon^3}\right)$ where $\widetilde{\text{ED}}_I$ has*

$$O_\varepsilon\left(n \cdot \left[\frac{T_{\text{Dec}}(\log n)}{\log n} + \frac{T_{\text{Dec}}(\log\log n)}{\log\log n} + \log^2\log n \cdot L(\log n)L(\log\log n) + \mathsf{poly}(\log\log n)\right]\right)$$

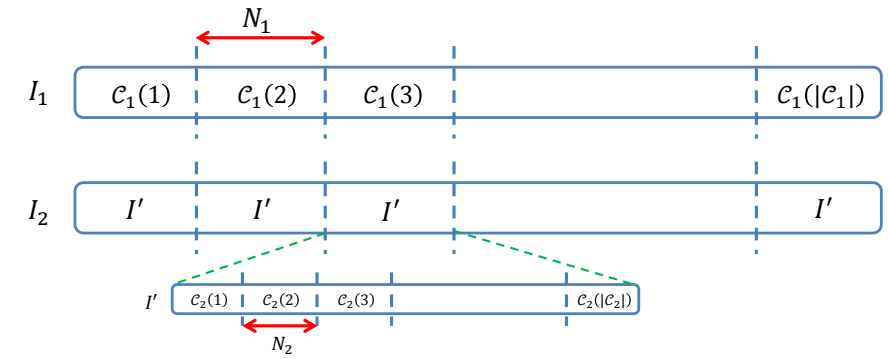
running time complexity. Further, if the given family of codes are efficiently encodable, I can be constructed in near-linear time.

These enhancements do not eventually yield an indexing scheme that works in $O(n \cdot \text{poly}(\log\log n))$ as the bottleneck of the indexing scheme's time complexity is the decoding time of the utilized list-recoverable code.

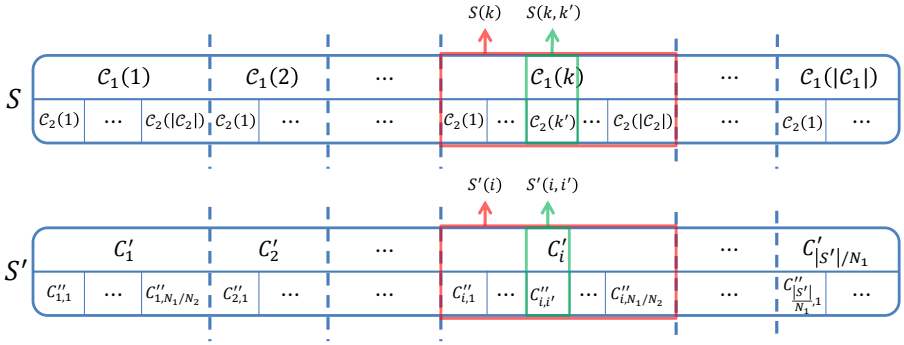
As of the time of writing this thesis, no deterministic list recoverable code with our ideal properties and a decoding time complexity faster than an unspecified large polynomial is found. However, because of the enhancements discussed in this section, improvements in decoding time complexity of list-recoverable codes can lead to ε-indexing schemes that run in $O(n \cdot \text{poly}(\log\log n))$ time. Particularly, having a linear-time $(\varepsilon, 1/\varepsilon, L(n) = \mathsf{poly}(\log n))$-list recoverable code would suffice.

9.5.1 Two Layer Indexing

Our enhanced indexing sequence I consists of the coordinate-wise concatenation of two string I_1 and I_2 where I_1 is the ordinary indexing sequence as described in Section 9.4,



(a) Construction of enhanced indexing string.



(b) Decoding for enhanced construction.

Figure 9.2

i.e, the codewords of a code $\mathcal{C}_1$ with block length N_1, and I_2 is repetitions of an ordinary indexing sequence I' of length N_1 constructed using some code $\mathcal{C}_2$. (See Fig. 9.2a)

In other words, let $\mathcal{C}_2$ be a code of block length N_2 and rate r_2 over alphabet $\Sigma_{\mathcal{C}_2}$ that is L_2-list decodable from $N_2(1-\varepsilon)$ insertions and $N_2(1-\varepsilon)$ deletions. Writing the codewords of $\mathcal{C}_2$ back to back would give the string I' of length $|I'| = N_2 \cdot |\Sigma_{\mathcal{C}_2}|^{N_2 r_2}$. Then, let code $\mathcal{C}_1$ be a code of block length $N_1 = |I'|$ and rate r_1 over alphabet $\Sigma_{\mathcal{C}_1}$ that is L_1-list decodable from $N_1(1-\varepsilon)$ insertions and $N_1(1-\varepsilon)$ deletions. We form string I_1 by writing the codewords of $\mathcal{C}_1$ one after another and string I_2 by repeating I' for $|\mathcal{C}_1|$ times. Finally, $I = (I_1, I_2)$.

We provide a decoding algorithm for indexing sequence I that is very similar to Algorithm 11 with an extra step in the construction of bipartite graph G that reduces the number of edges at the cost of a weaker yet still constant approximation guarantee.

In Line 8 of Algorithm 11, instead of adding an edge between any two pair of identical symbols in $S(k)$ and $S'(i)$ (that can be as many as $\log^2 n$), the algorithm runs another level of list-decoding and window-limiting based on the copy of I' that is a component of $S(k)$. In other word, the algorithm uses the decoder of $\mathcal{C}_2$ for any sub-block of length N_2 in $S'(i)$, like $S'(i,i')$, to find up to L_2 sub-blocks of length N_2 in $S(k)$, like $S(k,k')$, and adds an edge between any two identical symbols between $S'(i,i')$ and $S(k,k')$. We denote the portion of $S'(i,i')$ that corresponds to $\mathcal{C}_2$ codewords by $C''_{i,i'}$. (See Fig. 9.2b) A formal description is available in Algorithm 12.

Algorithm 12 $(1+23\varepsilon)$-Approximation for Edit Distance

1: **procedure** ENHANCED-ED-APPROX($S, S', N_1, N_2, \mathrm{Dec}_{\mathcal{C}_1}(\cdot), \mathrm{Dec}_{\mathcal{C}_2}(\cdot)$)
2: Make empty bipartite graph G with parts of size $|S|$ and $|S'|$
3: $w = \frac{1}{\varepsilon}$
4: **for each** $S'(i) = C'_i \times \left[C''_{i,1}, C''_{i,2}, \cdots, C''_{i,N_1/N_2}\right] \times m'[N_1(i-1), N_1 i - 1]$ **do**
5: $List_1 \leftarrow \mathrm{Dec}_{\mathcal{C}_1}(C'_i)$
6: **for each** $j \in List_1$ **do**
7: **for** $k \in [j-w, j+w]$ **do**
8: **for** $i' \in [1, N_1/N_2]$ **do**
9: $List_2 \leftarrow \mathrm{Dec}_{\mathcal{C}_2}(C''_{i,i'})$
10: **for each** $j' \in List_2$ **do**
11: **for** $k' \in [j'-w, j'+w]$ **do**
12: Connect any pair of vertices in G that correspond to identical symbols in $S(k,k')$ and $S'(i,i')$.
13: $\mathcal{M}_{ALG} \leftarrow$ Largest non-crossing matching in G (Using Theorem 9.3.6)
14: **return** $\mathcal{M}_{ALG}$

Theorem 9.5.2. *Algorithm 12 runs in* $O\left(\frac{n}{N_1} \cdot T_{\mathrm{Dec}_{\mathcal{C}_1}}(N_1) + \frac{n}{N_2} \cdot T_{\mathrm{Dec}_{\mathcal{C}_2}}(N_2) + \frac{N_2 L_1 L_2}{\varepsilon^2} \cdot n \log\log n\right)$ *time for* $n = \max(|S|, |S'|)$.

Proof. The algorithm uses the decoder of $\mathcal{C}_1$, $\frac{n}{N_1}$ times and the decoder of $\mathcal{C}_2$, $\frac{n}{N_2}$ times. G can have up to $\frac{n}{N} \cdot \frac{L_1}{\varepsilon} \cdot \frac{N_1}{N_2} \cdot \frac{L_2}{\varepsilon} \cdot N_2^2 = \frac{N_2 L_1 L_2}{\varepsilon^2} \cdot n$ edges. Therefore, the use of Hunt and

Szymanski's [HS77] algorithm (Theorem 9.3.6) will take $O\left(N_2 L_1 L_2/\varepsilon^2 \cdot n \log\log n\right)$ time. Therefore, the time complexity is as claimed. □

Theorem 9.5.3. *For $0 < \varepsilon < \frac{1}{121}$, Algorithm 12 computes a set of up to $(1+23\varepsilon)\cdot \mathrm{ED}(S, S')$ insertions and deletions that turn S into S'.*

Proof. In the proof of Theorem 9.4.4, we proved that for the graph G in Algorithm 11, $\mathrm{ED}_{ALG} \leq (1+11\varepsilon)\mathrm{ED}(S, S')$. In other words, the number of unmatched vertices in the largest non-crossing matching in that graph is at most $(1+11\varepsilon)$ times the number of unmatched vertices in the bipartite graph that corresponds to the longest common subsequence between S and S'.

As graph G in Algorithm 12 is the same as the one in Algorithm 11 with some extra edges removed, we only need to show that removing the extra edges does not increase the number of non-matched vertices in the largest non-crossing matching by more than a $(1+O(\varepsilon))$ multiplicative factor. This can be directly concluded from Theorem 9.4.4 since the extra removed edges are eliminated by doing the same procedure between pairs of codewords of $\mathcal{C}_2$ that is done between the strings in the statement of Theorem 9.4.4. In fact, using similar budget-based arguments as in Eqs. (9.1) and (9.2), the extra edge removal step will only increase the edit distance by a $(1+11\varepsilon)$ factor. This leads to the following upper bound on the approximation ratio of Algorithm 12 that holds for $\varepsilon < \frac{1}{121}$.

$$(1+11\varepsilon)(1+11\varepsilon)\mathrm{ED}(S, S') \leq (1+23\varepsilon)\mathrm{ED}(S, S')$$

□

9.5.2 Proof of Theorem 9.5.1

Proof. Let $\varepsilon' = \varepsilon/46$. Thus, the given family of codes is $\left(\varepsilon', \frac{6}{\varepsilon'}, L(\cdot)\right)$-list recoverable.

Take the code $\mathcal{C}_1$ as a code with block length N_1 from the given family of codes where N_1 is large enough so that $N_1 \cdot |\Sigma|^{r/2\cdot N_1} \geq n$. Similarly, take $\mathcal{C}_2$ with block length N_2 so that $N_2 \cdot |\Sigma|^{r/2\cdot N_2} \geq N_1$. For a large enough n, rates of $\mathcal{C}_1$ and $\mathcal{C}_2$ are at least $r/2$. We reduce the rates of $\mathcal{C}_1$ and $\mathcal{C}_2$ to $r/2$ by arbitrarily removing codewords from them.

We now use Theorem 9.7.4 with parameters $\varepsilon_{\text{conv}} = \varepsilon'$ and $\gamma_{\text{conv}} = 1-2\varepsilon'$ to convert list-recoverable codes $\mathcal{C}_1$ and $\mathcal{C}_2$ to list-decodable insertion-deletion codes $\tilde{\mathcal{C}}_1$ and $\tilde{\mathcal{C}}_2$ by indexing their codewords with appropriately chosen indexing sequences from Theorem 9.2.3 and synchronization strings. Note that we can do this conversion using Theorem 9.7.4 since $\gamma_{\text{conv}} = 1-2\varepsilon' \leq \frac{l_{\mathcal{C}_i}\cdot\varepsilon_{\text{conv}}}{3} - 1 = \frac{6/\varepsilon'\cdot\varepsilon'}{3} - 1 = 1$. Also, $\tilde{\mathcal{C}}_i$ can $L(N_i)$-list decode from any $\gamma_{\text{conv}} = 1-2\varepsilon'$ fraction of insertions and any $1-\alpha_{\mathcal{C}_i} - \varepsilon_{\text{conv}} = 1 - \frac{\varepsilon}{46} - \varepsilon' = 1-2\varepsilon'$ fraction of deletions in $T_{\text{Dec}}(N_i) + O\left(N_i \mathsf{poly}(\log N_i)\right)$.

Also, it is known how to construct ε_s-synchronization strings and ε_I-indexing schemes needed in Theorem 9.7.4. ε_s-synchronization strings can be constructed in linear time in terms of their length over an alphabet of size $\varepsilon_s^{-O(1)}$ and ε_I-indexing sequences from Theorem 9.2.3 can be constructed in near-linear time over an alphabet of size $\exp\left(\frac{\log(1/\varepsilon_I)}{\varepsilon_I^3}\right)$. Therefore, the alphabets of $\tilde{\mathcal{C}}_1$ and $\tilde{\mathcal{C}}_2$ will be of size $|\Sigma| \times \exp\left(\frac{\log(1/\varepsilon')}{\varepsilon'^3}\right)$.

We now use codes $\tilde{\mathcal{C}}_1$ and $\tilde{\mathcal{C}}_2$ in the structure described in the beginning of Section 9.5.1 to obtain an indexing sequence I of length n. Since the conversion of each codeword of $\mathcal{C}_i$ to $\tilde{\mathcal{C}}_i$ consumes near-linear time in terms of N_i, if the codes $\mathcal{C}_i$ are efficiently encodable, string I can be constructed in near-linear time. Also, the above-mentioned discussion on alphabet sizes of $\tilde{\mathcal{C}}_i$ entails that I will be a string over an alphabet of size $|\Sigma|^2 \times \exp\left(\frac{\log(1/\varepsilon')}{\varepsilon'^3}\right)$.

We now have to provide an algorithm that produces a $(1+\varepsilon)$-approximation for the edit distance using I. In the same spirit as the algorithm provided in the proof of Theorem 9.5.1, we define algorithm $\widetilde{\mathrm{ED}}_I$ as an algorithm that takes $S \times I$ and S' and runs an instance of Algorithm 12 with $S \times l$, S', N_1, N_2, and decoders of $\tilde{\mathcal{C}}_i$ as its input.

As codes $\tilde{\mathcal{C}}_i$ list decode from $1 - 2\varepsilon'$ fraction of insertions and deletions, Theorem 9.5.3 guarantees that $\widetilde{\mathrm{ED}}_I$ generates a set of at most $(1+23\cdot 2(\varepsilon'))\mathrm{ED}(S\times I, S') = (1+\varepsilon)\mathrm{ED}(S\times I, S')$ insertions and deletions over $S \times I$ that converts it to S'.

Finally, since $N_1 = O(\log n)$, $N_2 = O(\log\log n)$ and $\tilde{\mathcal{C}}_i$ list decode in $T_{\mathrm{Dec}}(N_i) + O\left(N_i \mathsf{poly}(\log N_i)\right)$ time, Theorem 9.5.2 guarantees that $\widetilde{\mathrm{ED}}_I$ runs in

$$
\begin{aligned}
& O\left(\frac{n}{N_1}\cdot T_{\mathrm{Dec}_{\tilde{\mathcal{C}}_1}}(N_1) + \frac{n}{N_2}\cdot T_{\mathrm{Dec}_{\tilde{\mathcal{C}}_2}}(N_2) + \frac{N_2 L_1 L_2}{\varepsilon^2}\cdot n\log\log n\right) \\
= \; & O_\varepsilon\Bigg(\frac{n}{\log n}\cdot[T_{\mathrm{Dec}}(\log n) + \log n \mathsf{poly}(\log\log n)] + \\
& \quad \frac{n}{\log\log n}\cdot[T_{\mathrm{Dec}}(\log\log n) + \log\log n \mathsf{poly}(\log\log\log n)] + \\
& \quad n\log^2\log n \cdot L(\log n)L(\log\log n)\Bigg) \\
= \; & O_\varepsilon\left(n\cdot\left[\frac{T_{\mathrm{Dec}}(\log n)}{\log n} + \frac{T_{\mathrm{Dec}}(\log\log n)}{\log\log n} + \log^2\log n\cdot L(\log n)L(\log\log n) + \mathsf{poly}(\log\log n)\right]\right)
\end{aligned}
$$

time. □

9.6 Randomized Indexing

In this section, we will prove the following theorem by taking similar steps as in the proof of Theorem 9.5.1 to construct an indexing scheme according to the structure introduced in Section 9.5.1.

Theorem 9.6.1. *For any $\varepsilon_0 > 0$, $\varepsilon_1, \varepsilon_2 \in (0,1)$, and integer n, there exists a randomized indexing scheme $(I, \widetilde{\mathrm{ED}}_I)$ of length n where $\widetilde{\mathrm{ED}}_I(S\times I, S')$ runs in $O(n\log^{\varepsilon_0} n)$ time and proposes a set of insertions and deletions that turns $S\times I$ into S' and contains up to $(1+\varepsilon_1)\mathrm{ED}(S\times I, S') + \varepsilon_2|S'|$ operations with probability $1 - \frac{1}{n^{O(1)}}$.*

Note that, as opposed to the rest of the results in this chapter, Theorem 9.6.1 provides an approximation guarantee with both multiplicative and additive components.

To construct such an indexing scheme using the structure introduced in Section 9.5.1, we will use a list-decodable insertion-deletion code of block length $O(\log\log n)$ from Corollary 9.3.4 and use Theorem 9.7.4 to obtain a list-decodable insertion-deletion code of block length $O(\log n)$ from the following list recoverable codes of [HRZW19].

Theorem 9.6.2 (Corollary of Theorem 7.1 of Hemenway et al. [HRZW19])**.** *For any $\rho \in [0,1]$, $\varepsilon > 0$, and positive integer l, there exist constants q_0 and c_0 so that, for any $c < c_0$ and infinitely many integers $q \geq q_0$, there exists an infinite family of codes achieving the rate ρ over an alphabet Σ of size $|\Sigma| = q$ that is encodable in n^{1+c} time and probabilistically $(\rho + \varepsilon, l, L(n))$-list recoverable in n^{1+c} time with success probability 2/3 and $L(n) = O_{\varepsilon,\rho}(\exp(\exp(\exp(\log^* n))))$ where n denotes the block length.*

Before providing the proof of Theorem 9.6.1, we mention a couple of necessary lemmas.

Lemma 9.6.3. *Let $(\alpha, l, L(n))$-list-recoverable code $\mathcal{C}$ have a probabilistic decoder that runs in $T_{\text{Dec}}(n)$ and works with probability p. Then, for any integer k, $\mathcal{C}$ can be $(\alpha, l, k \cdot L(n))$-list-recovered in $kT_{\text{Dec}}(n)$ time with $1 - (1-p)^k$ success probability.*

Proof. Use a decoding procedure for $\mathcal{C}$ that repeats the given decoder k times and outputs the union of the lists produced by them. The final list size will be at most $kL(n)$ long, the running time will be $kT_{\text{Dec}}(n)$, and the failure probability, i.e., the probability of the output list not containing the correct codeword is at most $(1-p)^k$. □

Another required ingredient to the proof of Theorem 9.6.1 is to show how a probabilistic decoder affect the approximation guarantee of Algorithm 12. To this end, we provide the following lemma as an analogy of Theorem 9.5.3 when the decoder of code $\mathcal{C}_1$ is not deterministic.

Lemma 9.6.4. *Let the decoder of code $\mathcal{C}_1$ ($\text{Dec}_{\mathcal{C}_1}(\cdot)$) be a randomized algorithm that L_1-list decodes the code $\mathcal{C}_1$ with probability $1-p$. Then, with probability $1 - e^{-\frac{2|S'|p}{3N_1}}$, Algorithm 12 will generate a set of up to $(1+23\varepsilon)\text{ED}(S,S') + 2p|S'|$ insertions and deletions that turn S into S'.*

Proof. If $\text{Dec}_{\mathcal{C}_1}(\cdot)$ worked with probability 1, the outcome of $\mathcal{A}$ would contain up to $(1 + 23\varepsilon_1)$ insertions and deletions. Each time that $\text{Dec}_{\mathcal{C}_1}$ fails to correctly list-decode a block of length N_1 (C'_i), up to N_1 edges from $\mathcal{M}_{ALG}$ might be lost and, consequently, there can be up to $2N_1$ units of increase in the number of insertions and deletions generated by $\mathcal{A}$.

There are a total of $n = |S'|/N_1$ list decodings and each might fail with probability p. Using the Chernoff bound,

$$\Pr(\text{more than } 2np \text{ failures}) \leq e^{-2np/3} = e^{-\frac{2|S'|p}{3N_1}}.$$

Thus, with probability $1 - e^{-\frac{2|S'|p}{3N_1}}$, the output of $\mathcal{A}$ contains $(1+23\varepsilon)\text{ED}(S,S') + 2npN_1 = (1+23\varepsilon)\text{ED}(S,S') + 2p|S'|$ or less insertions and deletions. □

We are now adequately equipped to prove Theorem 9.6.1.

9.6.1 Proof of Theorem 9.6.1

Proof. Our construction closely follows the steps taken in the proof of Theorem 9.5.1. Let $\varepsilon' = \varepsilon_1/46$. Take $\mathcal{C}_1$ from the Theorem 9.6.2 with parameters $\varepsilon_{\mathcal{C}_1} = \varepsilon'$, $\rho_{\mathcal{C}_1} = 2\varepsilon'$, $l_{\mathcal{C}_1} = 6/\varepsilon'$, $c_{\mathcal{C}_1} = \varepsilon_0$, and block length N_1 where N_1 is large enough so that $N_1 \cdot q_{\mathcal{C}_1}^{\rho_{\mathcal{C}_1}/2\cdot N_1} \geq n$ where $q_{\mathcal{C}_1}$ is the size of the alphabet of the family codes.

According to Theorem 9.6.2, $\mathcal{C}_1$ is probabilistically $(\varepsilon', \varepsilon'/6, L(N_1))$-list recoverable in $O_{\varepsilon_1}(N_1^{1+\varepsilon_0})$ time where $L(N_1) = \exp(\exp(\exp(\log^* N_1)))$ and success probability is $2/3$. We use Lemma 9.6.3 with repetition number parameter $k = \log_3 \frac{2}{\varepsilon_2}$ to obtain a $\left(\varepsilon', \frac{\varepsilon'}{6}, O\left(\log \frac{1}{\varepsilon_2} L(N_1)\right)\right)$-list recovery algorithm for $\mathcal{C}_1$ that succeeds with probability $1 - (\frac{1}{3})^k = 1 - \frac{\varepsilon_2}{2}$ and runs in $O_{\varepsilon_1}\left(\frac{N_1^{1+\varepsilon_0}}{\varepsilon_2}\right)$ time.

We now use Theorem 9.7.4 with parameters $\varepsilon_{\text{conv}} = \varepsilon'$ and $\gamma_{\text{conv}} = 1-2\varepsilon'$ to convert list-recoverable code $\mathcal{C}_1$ to a list-decodable insertion-deletion code $\tilde{\mathcal{C}_1}$ by indexing its codewords with an appropriately chosen indexing sequence from Theorem 9.2.3 and a synchronization string. Note that we can do this conversion using Theorem 9.7.4 since $\gamma_{\text{conv}} = 1 - 2\varepsilon' \leq \frac{l_{\mathcal{C}_1}\cdot\varepsilon_{\text{conv}}}{3} - 1 = \frac{6/\varepsilon'\cdot\varepsilon'}{3} - 1 = 1$. Also, $\tilde{\mathcal{C}_1}$ can $O\left(\log \frac{1}{\varepsilon_2} L(N_1)\right)$-list decode from any $\gamma_{\text{conv}} = 1-2\varepsilon'$ fraction of insertions and any $1 - \alpha_{\mathcal{C}_1} - \varepsilon_{\text{conv}} = 1 - \frac{\varepsilon}{46} - \varepsilon' = 1 - 2\varepsilon'$ fraction of deletions in $O_{\varepsilon_1,\varepsilon_2}\left(N_1^{1+\varepsilon_0} + N_1 \mathsf{poly}(\log N_1)\right)$.

We further take code $\tilde{\mathcal{C}_2}$ from Corollary 9.3.4 with parameter $\varepsilon_{\tilde{\mathcal{C}_2}} = 2\varepsilon'$ and block length N_2 large enough so that $N_2 \cdot q_{\tilde{\mathcal{C}_2}}^{\varepsilon'/2\cdot N_2} \geq N_1$. $\tilde{\mathcal{C}_2}$ is $\exp(\exp(\exp(N_2)))$-list decodable from any $1 - 2\varepsilon'$ fraction of insertions and $1 - 2\varepsilon'$ fraction of deletions.

String I for the indexing scheme is constructed according to the structure described in Section 9.5.1 using $\tilde{\mathcal{C}_1}$ and $\tilde{\mathcal{C}_2}$.

We define algorithm $\widetilde{\text{ED}}_I$ as an algorithm that takes $S \times I$ and S' and runs an instance of Algorithm 12 with $S \times I$, S', N_1, N_2, and decoders of $\tilde{\mathcal{C}_i}$ as its input. As codes $\tilde{\mathcal{C}_i}$ list decode from $1 - 2\varepsilon'$ fraction of insertions and deletions, Lemma 9.6.4 guarantees that $\widetilde{\text{ED}}_I$ generates a set of insertions and deletions over $S \times I$ that converts it to S' and is of size $(1 + 23 \cdot 2\varepsilon')\text{ED}(S \times I, S') + 2 \cdot \frac{\varepsilon_2}{2}|S'| = (1+\varepsilon)\text{ED}(S \times I, S') + \varepsilon_2|S'|$ or less with probability $1 - e^{-\frac{\varepsilon_2}{3N_1}} = 1 - e^{-O\left(\frac{\varepsilon_2}{\log n}\right)} = 1 - \frac{1}{n^{O_{\varepsilon_1,\varepsilon_2}(1)}}$.

Finally, since $N_1 = O(\log n)$, $N_2 = O(\log\log n)$, $\tilde{\mathcal{C}_1}$ is list-decodable in $O(N_1^{1+\varepsilon_0} + N_1\mathsf{poly}(\log N_1))$ time and $\tilde{\mathcal{C}_2}$ is efficiently list-decodable, Theorem 9.5.2 guarantees that

$\widetilde{\mathrm{ED}}_I$ runs in

$$\begin{aligned}
& O_{\varepsilon_1,\varepsilon_2}\left(\frac{n}{N_1}\cdot T_{\mathrm{Dec}_{\tilde{\mathcal{C}}_1}}(N_1)+\frac{n}{N_2}\cdot T_{\mathrm{Dec}_{\tilde{\mathcal{C}}_2}}(N_2)+N_2L_1L_2\cdot n\log\log n\right)\\
= \; & O_{\varepsilon_1,\varepsilon_2}\left(\frac{n}{\log n}\cdot T_{\mathrm{Dec}_{\tilde{\mathcal{C}}_1}}(\log n)+\frac{n}{\log\log n}\cdot T_{\mathrm{Dec}_{\tilde{\mathcal{C}}_2}}(\log\log n)+n\log^2\log n L_{\tilde{\mathcal{C}}_1}(N_1)L_{\tilde{\mathcal{C}}_2}(N_2)\right)\\
= \; & O_{\varepsilon_1,\varepsilon_2}\Bigg(\frac{n}{\log n}\cdot\left[\log^{1+\varepsilon_0}n+\log n\mathsf{poly}(\log\log n)\right]+\frac{n}{\log\log n}\cdot[\mathsf{poly}(\log\log n)]+\\
& \qquad n\log^2\log n\cdot\exp(\exp(\exp(\log^* n)))\Bigg)\\
= \; & O_{\varepsilon_1,\varepsilon_2}\left(n\log^{\varepsilon_0}n\right)
\end{aligned}$$

time. □

9.7 Near-Linear Time Insertion-Deletion Codes

The construction of efficient (uniquely-decodable) insertion-deletion codes from Chapter 3 and list-decodable codes from Chapter 4 profoundly depend on decoding synchronization strings that are attached to codewords of an appropriately chosen Hamming-type code. The decoding procedure, which was introduced in Chapter 3, consists of multiple rounds of computing the longest common subsequence (LCS) between a synchronization string and a given string. In this section, we will show that using the indexing schemes that are introduced in this chapter, one can compute approximations of the LCSs instead of exact LCSs to construct insertion-deletion codes of similar guarantees as in Chapters 3 and 4 that have faster decoding complexity.

Specifically, for uniquely-decodable insertion-deletion codes, Chapter 3 provided codes with linear encoding-time and quadratic decoding-time that can approach the singleton bound, i.e., for any $0 < \delta < 1$ and $0 < \varepsilon < 1 - \delta$ can correct from δ-fraction of insertions and deletions and achieve a rate of $1-\delta-\varepsilon$. Further, Chapter 8 provided codes with linear encoding complexity and near-linear decoding complexity can can correct from $\delta < 1/3$ fraction of insertions and deletions but only achieve a rate of $1 - 3\delta - \varepsilon$. In Theorem 9.2.4 we will provide insertion-deletion codes that give the best of the two worlds, i.e., approach the Singleton bound and can be decoded in near-linear time.

Further, in Theorem 9.7.4, we show that the same improvement can be made over list-decodable insertion-deletion codes of Chapter 4. However, this improvement brings downs the complexity of all components of the decoding procedure to near-linear time except the part that depends on the decoding of a list-recoverable code that is used as a black-box in the construction from Chapter 4. Even though this progress does not immediately improve the decoding time of list-decodable codes of Chapter 4, it opens the door to enhancement of the decoding complexity down to potentially a near-linear time by the future advances in the design of list-recoverable codes.

9.7.1 Enhanced Decoding of Synchronization Strings via Indexing

Let S be an ε-synchronization string that is communicated through a channel that suffers from a certain fraction of insertions and deletions. A decoding algorithm Dec_S for synchronization string S under such channel is an algorithm that takes string that is arrived at the receiving end of the channel, and for each symbol of that string, guesses its actual position in S. We measure the quality of the decoding algorithm Dec_S by a metric named as *misdecodings*. A misdecoding in the above-mentioned decoding procedure is a symbol of S that (1) is not deleted by the channel *and* (2) is not decoded correctly by Dec_S. (find formal definitions in Chapter 3)

The important quality of synchronization strings that is used in the design of insertion-deletion codes in Chapters 3 and 4 is that there are decoders for any ε-synchronization string that run in quadratic time $O(n^2/\varepsilon)$ and guarantee $O(n\sqrt{\varepsilon})$ misdecodings. In this chapter, by indexing synchronization strings with indexing sequences introduced in Theorem 9.2.3, we will show that one can obtain a near-linear decoding that provides similar misdecoding guarantee.

In the rest of this section, we first present and prove a theorem that shows an indexed synchronization string can be decoded in near-linear time with guarantees that are expected in Theorem 3.5.14 and Lemma 4.3.2. We then verify that the steps taken in Chapters 3 and 4 still follow through.

Theorem 9.7.1. *Let S be a string of length n that consists of the coordinate-wise concatenation of an ε_s-synchronization string and an ε_I-indexing sequence from Theorem 9.2.3. Assume that S goes through a channel that might impose up to $\delta \cdot n$ deletions and $\gamma \cdot n$ symbol insertions on S for some $0 \leq \delta < 1$ and $0 \leq \gamma$ and arrives as S' on the receiving end of the channel. For any positive integer K, there exists a decoding for S' that runs in $O(Kn\mathsf{poly}(\log n))$ time, guarantees up to $n\left(\frac{1+\gamma}{K(1+\varepsilon_I)} + \frac{\varepsilon_I(1+\gamma/2)}{1+\varepsilon_I} + K\varepsilon_s\right)$ misdecodings, and does not decode more than K received symbol to any number in $[1, n]$.*

Before proceeding to the proof of Theorem 9.7.1, we present and prove the following simple yet useful lemma.

Lemma 9.7.2. *Let us have a set of insertions and deletions that converts string S_1 to string S_2 which is of size $\text{ED}_{APP} \leq (1+\varepsilon)\text{ED}(S_1, S_2)$. The common subsequence between S_1 and S_2 that is implied by such a set (LCS_{APP}) is of size $(1+\varepsilon)|\text{LCS}| - \frac{\varepsilon}{2}(|S_1| + |S_2|)$ or larger.*

Proof.

$$\begin{aligned}
|\text{LCS}_{APP}| &= \frac{|S_1| + |S_2| - \text{ED}_{APP}}{2} \\
&\geq \frac{|S_1| + |S_2| - (1+\varepsilon)\text{ED}(S_1, S_2)}{2} \\
&= \frac{|S_1| + |S_2| - (1+\varepsilon)(|S_1| + |S_2| - 2|\text{LCS}|)}{2} \\
&= (1+\varepsilon)|\text{LCS}| - \frac{\varepsilon}{2}(|S_1| + |S_2|)
\end{aligned}$$

□

Proof of Theorem 9.7.1. The global repositioning algorithm introduced in Chapter 3 and used in Chapter 4, consists of K repetitions of the following steps:

1. Find the longest common subsequence (LCS) of S and S'.
2. For any pair $(S[i], S'[j])$ in the LCS, decode $S'[j]$ as ith sent symbol.
3. Remove all members of the LCS from S' (not in S).

Finally, the algorithm declares a special symbol $\perp$ as the decoded position of all elements of S' that are not included in any of the K LCSs.

To derive a decoding algorithm as promised in the statement of this lemma, we implement similar steps except we make use of the indexing scheme and compute an approximation of LCS instead of the LCS itself. This crucial step reduces the quadratic time required in the global repositioning from Chapter 3 to near-linear time.

In Chapter 3, it has been shown that any assignment from Item 2 that is derived from any common subsequence between S and S' (not necessarily a LCS) does not contain more than $n\varepsilon_s$ misdecodings, i.e., successfully transmitted symbols of S that are decode incorrectly. (see Theorem 9.3.1). Therefore, after K repetitions, among symbols of S that are not deleted, there are at most $Kn\varepsilon_s$ ones that are decoded incorrectly.

To find an upper bound for the misdecodings of this algorithm, we need to bound above the number of successfully transmitted symbols that are not included in any LCS, i.e., decoded as $\perp$ as well. Let r be number of successfully transmitted symbols of S that remain undecoded after K repetitions of the matching procedure described above. Note that these symbols form a LCS of length r between S and the remainder of S' after all symbol eliminations throughout K repetitions. Indeed, this implies that the size of the LCS at the beginning of each repetition is at least r. Therefore, by Lemma 9.7.2, the size of the approximate longest common sequence found in each matching is at least $(1+\varepsilon_I)r - \varepsilon_I/2(|S| + |S'|) \geq (1+\varepsilon_I)r - \varepsilon_I n(1+\gamma/2)$. Note that sum of the size of all K common subsequences plus the remaining vertices cannot exceed $|S'| \leq (1+\gamma)n$. Therefore,

$$\begin{aligned} & K \cdot [(1+\varepsilon_I)r - \varepsilon_I n(1+\gamma/2)] \leq (1+\gamma)n \\ \Rightarrow \quad & r \leq n \cdot \left[\frac{1+\gamma}{K(1+\varepsilon_I)} + \frac{\varepsilon_I(1+\gamma/2)}{1+\varepsilon_I}\right] \end{aligned} \tag{9.4}$$

Using (9.4) along with the fact that there are at most $Kn\varepsilon_s$ incorrectly decoded symbols of S gives that the overall number of misdecodings is at most $n \cdot \left[\frac{1+\gamma}{K(1+\varepsilon_I)} + \frac{\varepsilon_I(1+\gamma/2)}{1+\varepsilon_I} + K\varepsilon_s\right]$.

Further, as algorithm consists of K computations of the approximated longest common subsequence as described in Section 9.4, the running time complexity is $O(Kn\mathsf{poly}(\log n))$.

Finally, note that in each of the K rounds, there is at most one element that gets decoded as each number in $[1, n]$. Therefore, throughout the course of the algorithm, for each $i \in [1, n]$, there are at most K elements of S' that are decoded as i. □

9.7.2 Near-Linear Time (Uniquely-Decodable) Insertion-Deletion Codes

The construction of Singleton-bound-approaching uniquely-decodable insertion-deletion codes from Chapter 3 is consisted of a Singleton approaching error correcting code and a synchronization string. More precisely, for a given δ and ε and a sufficiently large n, we took a synchronization string S of length n and a Singleton-bound-approaching error correcting code $\mathcal{C}$ with block length n (from [GI05]) and indexes each codeword of $\mathcal{C}$, symbol by symbol, with symbols of S. If S is over alphabet Σ_S and $\mathcal{C}$ is over alphabet $\Sigma_{\mathcal{C}}$, the resulting code would be over $\Sigma_{\mathcal{C}} \times \Sigma_S$.

As for the decoding procedure, note that the input of the decoder is some code word of $\mathcal{C}$, indexed with S, that might be altered by up to $\delta \cdot n$ insertions and deletions. Such insertions and deletions might remove some symbols, adds some new ones, or shift the position of some of them. The decoder uses the synchronization portion of each symbol to guess its actual position (in the codeword prior to $n \cdot \delta$ insertions and deletions) and then uses the decoder of code $\mathcal{C}$ to figure out the sent codeword.

Before proceeding to the proof of Theorem 9.2.4, we represent the following useful theorem from Chapter 3.

Theorem 9.7.3 (Theorem 3.3.2)**.** *Given a synchronization string S over alphabet Σ_S, an (efficient) decoding algorithm $\mathcal{D}_S$ with at most k misdecodings and decoding complexity $T_{\mathcal{D}_S}(n)$ and an (efficient) ECC $\mathcal{C}$ over alphabet $\Sigma_{\mathcal{C}}$ with rate $R_{\mathcal{C}}$, encoding complexity $T_{\mathcal{E}_{\mathcal{C}}}$, and decoding complexity $T_{\mathcal{D}_{\mathcal{C}}}$ that corrects up to $n\delta + 2k$ half-errors, one obtains an insdel code that can be (efficiently) decoded from up to $n\delta$ insertions and deletions. The rate of this code is at least*

$$\frac{R_{\mathcal{C}}}{1 + \frac{\log|\Sigma_S|}{\log|\Sigma_{\mathcal{C}}|}}.$$

The encoding complexity remains $T_{\mathcal{E}_{\mathcal{C}}}$, the decoding complexity is $T_{\mathcal{D}_{\mathcal{C}}} + T_{\mathcal{D}_S}(n)$ and the complexity of constructing the code is the complexity of constructing $\mathcal{C}$ and S.

We make use of Theorem 9.7.3 along with Theorem 9.7.1 to prove Theorem 9.2.4.

Proof of Theorem 9.2.4. As described earlier in this section, we construct this code by taking an error correcting code that approaches the Singleton bound and then index its codewords with symbols of an ε_s-synchronization string and an indexing scheme from Theorem 9.2.3 with parameter ε_I. For a given δ and ε, we choose $\varepsilon_I = \frac{\varepsilon}{18}$, $\varepsilon_s = \frac{\varepsilon^2}{288}$. Furthermore, we use the decoding algorithm from Theorem 9.7.1 with repetition parameter $K = \frac{24}{\varepsilon}$. With $\varepsilon_s, \varepsilon_I$, and K chosen as such, the decoding algorithm guarantees a misdecoding count of $n \cdot \left[\frac{1+\gamma}{K(1+\varepsilon_I)} + \frac{\varepsilon_I(1+\gamma/2)}{1+\varepsilon_I} + K\varepsilon_s\right] \leq n \cdot \left[\frac{\varepsilon}{12} + \frac{\varepsilon}{12} + \frac{\varepsilon}{12}\right] = \frac{n\varepsilon}{4}$ or less. (note that there can be up to δn insertions, i.e., $\gamma \leq \delta < 1$)

It has been shown in Chapter 8 that such synchronization string can be constructed in linear time over an alphabet of size $\varepsilon_s^{-O(1)}$. Also, the indexing sequence from Theorem 9.2.3 has an alphabet of size $\exp\left(\frac{\log(1/\varepsilon_I)}{\varepsilon_I^3}\right)$. Therefore, the alphabet size of the coordinate-

wise concatenation of the ε_s-synchronization string and the indexing sequence is $|\Sigma_S| = \exp\left(\frac{\log(1/\varepsilon)}{\varepsilon^3}\right)$.

As the next step, we take code $\mathcal{C}$ from [GI05] as a code with distance $\delta_{\mathcal{C}} = \delta + \frac{\varepsilon}{2}$ and rate $1 - \delta_{\mathcal{C}} - \frac{\varepsilon}{4}$ over an alphabet of size $|\Sigma_{\mathcal{C}}| = |\Sigma_S|^{4/\varepsilon}$. Note that $|\Sigma_S| = \exp\left(\frac{\log(1/\varepsilon)}{\varepsilon^3}\right)$, therefore, the choice of $|\Sigma_{\mathcal{C}}|$ is large enough to satisfy the requirements of [GI05]. $\mathcal{C}$ is also encodable and decodable in linear time.

Plugging $\mathcal{C}$ and S as described above in Theorem 9.7.3 gives an insertion-deletion code that can be encoded in linear time, be decoded in $O(Kn\mathsf{poly}(\log n))$ time, corrects from any δn insertions and deletions, achieves a rate of $\frac{R_{\mathcal{C}}}{1+\frac{\log|\Sigma_S|}{\log|\Sigma_{\mathcal{C}}|}} \geq \frac{1-\delta-3\varepsilon/4}{1+\varepsilon/4} \geq 1 - \delta - \varepsilon$, and is over an alphabet of size $\exp\left(\frac{\log(1/\varepsilon)}{\varepsilon^4}\right)$. □

9.7.3 Improved List-Decodable Insertion-Deletion Codes

A very similar improvement is also applicable to the design of list-decodable insertion-deletion codes from Chapter 4 as it also utilizes indexed synchronization strings and a similar position recovery procedure. In the following theorem, we will provide a black-box conversion of a given list-recoverable code to a list-decodable insertion-deletion code that only adds a near-linear time overhead to the decoding complexity. Hence, the following theorem paves the way to obtaining insertion-deletion codes that are list-decodable in near-linear time upon the design of near-linear time list-recoverable codes. We will use the following theorem to prove Theorem 9.2.5 at the end of this section.

Theorem 9.7.4. *Let $\mathcal{C} : \Sigma^{nR} \to \Sigma^n$ be a (α, l, L)-list recoverable code with rate R, encoding complexity $T_{Enc}(\cdot)$ and decoding complexity complexity $T_{Dec}(\cdot)$. For any $\varepsilon > 0$ and $\gamma \leq \frac{l\varepsilon}{3} - 1$, by indexing codewords of $\mathcal{C}$ with an $\varepsilon_s = \frac{\varepsilon^2}{9(1+\gamma)}$-synchronization string over alphabet Σ_s and $\varepsilon_I = \frac{\varepsilon}{3(1+\gamma/2)}$-indexing sequence over alphabet Σ_I, one can obtain an L-list decodable insertion-deletion code $\mathcal{C}' : \Sigma^{nR} \to [\Sigma \times \Sigma_s \times \Sigma_I]^n$ that corrects from $\delta < 1 - \alpha - \varepsilon$ fraction of deletions and γ fraction of insertions. $\mathcal{C}'$ is encodable and decodable in $O(T_{Enc}(n) + n)$ and $O_{\varepsilon,\gamma}(T_{Dec}(n) + n\mathsf{poly}(\log n))$ time respectively.*

Proof. We closely follow the proof of Theorem 4.3.1 except that we use an indexed synchronization string to speed up the decoding procedure.

Index the code $\mathcal{C}$ with an $\varepsilon_s = \frac{\varepsilon^2}{9(1+\gamma)}$-synchronization string and an $\varepsilon_I = \frac{\varepsilon}{3(1+\gamma/2)}$-indexing sequence as constructed in Theorem 9.2.3 to obtain code $\mathcal{C}'$.

In the decoding procedure, for a given word $\tilde{x}$ that is δn deletions and γn insertions far from some codeword $x \in \mathcal{C}'$, we first use the decoding algorithm from Theorem 9.7.1 to decode the index portion of symbols with parameter $K = \frac{3(1+\gamma)}{\varepsilon}$. This will give a list of up to $K = \frac{3(1+\gamma)}{\varepsilon} \leq l$ candidate symbols for each position of the codeword x.

We know from Theorem 9.7.1 that all but

$$n\left(\frac{1+\gamma}{K(1+\varepsilon_I)} + \frac{\varepsilon_I(1+\gamma/2)}{1+\varepsilon_I} + K\varepsilon_s\right) \leq n\left(\frac{\varepsilon}{3(1+\varepsilon_I)} + \frac{\varepsilon}{3(1+\varepsilon_I)} + \frac{\varepsilon}{3}\right) \leq n\varepsilon$$

of the symbols of x that are not deleted are in the correct list. As there are up to $n(1-\delta)$ deleted symbols, all but $n(1-\delta-\varepsilon) > n\alpha$ of the lists contain the symbol from the corresponding position in x. Having such lists, the receiver can use the list-recovery function of $\mathcal{C}$ to obtain an L-list-decoding for $\mathcal{C}'$.

The encoding complexity follows from the fact that synchronization strings be constructed in linear time (see Chapter 8), the decoding complexity follows from Theorem 9.7.1, and the alphabet of $\mathcal{C}'$ is trivially $\Sigma \times \Sigma_s \times \Sigma_I$ as it is obtained by indexing codewords of $\mathcal{C}$ with the ε_s-synchronization string and the ε_I-indexing sequence. □

We now use Theorem 9.7.4 to prove Theorem 9.2.5.

Proof of Theorem 9.2.5. Take list-recoverable code $\mathcal{C}$ from Theorem 9.6.2 with parameters $\rho_{\mathcal{C}} = 1 - \delta - \frac{\varepsilon}{2}$, $\varepsilon_{\mathcal{C}} = \frac{\varepsilon}{4}$, $l_{\mathcal{C}} = \frac{12\gamma+4}{\varepsilon}$, and $c_{\mathcal{C}} = \varepsilon_0$ over an alphabet Σ of adequately large size $|\Sigma| \geq q_{0,\mathcal{C}}$ which we determine later. According to Theorem 9.6.2, $\mathcal{C}$ has a rate of $\rho_{\mathcal{C}}$ and a randomized $(\rho_{\mathcal{C}} + \varepsilon_{\mathcal{C}}, l_{\mathcal{C}}, L(n) = \exp(\exp(\exp(\log^* n))))$-list recovery that works in $O(n^{1+\varepsilon_0})$ time and succeeds with probability 2/3.

We plug code $\mathcal{C}$ into Theorem 9.7.4 with parameters $\varepsilon_{\text{conv}} = \frac{\varepsilon}{4}$ and $\gamma_{\text{conv}} = \gamma$ to obtain code $\mathcal{C}'$. We can do this because $\gamma_{\text{conv}} \leq \frac{l_{\mathcal{C}}\varepsilon_{conv}}{3} - 1$. According to Theorem 9.7.4, $\mathcal{C}'$ is $L(n)$-list decodable from $1 - \rho_{\mathcal{C}} - \varepsilon_{\mathcal{C}} - \varepsilon_{\text{conv}} = \delta$ fraction of deletions and γ fraction of insertions in $O(n^{1+\varepsilon_0})$. This list-decoding is correctly done if the list-recovery algorithm works correctly. Therefore, the list decoder succeeds with probability 2/3 or more.

Note that the ε_s-synchronization strings in Theorem 9.7.4 exist over alphabets of size $|\Sigma_s| = \varepsilon_s^{-O(1)}$ and ε_I-indexing sequence exist over alphabets of size $|\Sigma_I| = \exp\left(\frac{\log(1/\varepsilon_I)}{\varepsilon_I^3}\right)$. Therefore, if we take alphabet Σ large enough so that $|\Sigma| \geq \max\left\{|\Sigma_s \times \Sigma_I|^{2/\varepsilon}, q_{0,\mathcal{C}}\right\} = \max\left\{\exp\left(\frac{\log(1/\varepsilon)}{\varepsilon^4}\right), q_{0,\mathcal{C}}\right\} = O_{\varepsilon_0,\varepsilon,\gamma}(1)$ the rate of the resulting code will be

$$\frac{\rho_{\mathcal{C}}}{1 + \frac{\log(|\Sigma_S| \times |\Sigma_l|)}{\log|\Sigma|}} \geq \frac{1-\delta-\varepsilon/2}{1+\varepsilon/2} \geq 1 - \delta - \varepsilon.$$

Finally, the encoding and decoding complexities directly follow from Theorem 9.7.4. □

9.8 Approximating Levenshtein Distance in $\tilde{O}(n^{1+\varepsilon_t})$

Thus far in this chapter, we studied approximations for the edit distance defined as the smallest number of insertions and deletions the are needed to convert a string to another one. In this section, we focus on a similar notion celled *Levenshtein distance* (LevD). Generally, a symbol substitution, i.e., converting a symbol to another one, can be interpreted as a deletion followed by an insertion at the same position. This interpretation, however, associates two units of cost to the symbol substitution operation. Levenshtein Distance is a similar notion that assigns a unit cost to symbol substitutions as well as insertions and deletions. More precisely, Levenshtein distance between two strings is the minimum number of symbol insertions, deletions, or substitutions that are needed to convert a string to another one.

In this section, we show that indexing strings with an string of type used in the indexing scheme of Theorem 9.2.3 enables us to approximate Levenshtein distance within a $1+\varepsilon$ multiplicative factor in $\tilde{O}_\varepsilon(n^{1+\varepsilon_t})$ time for any $0 < \varepsilon, \varepsilon_t$. The following theorem summarizes the main result of this section.

Theorem 9.8.1. *For any $0 < \varepsilon, \varepsilon_t$, there exists a randomized indexing scheme for any length n $(I, \widetilde{\mathrm{LevD}}_I)$ where $\widetilde{\mathrm{LevD}}_I(S \times I, S')$ runs in $\tilde{O}_\varepsilon(n^{1+\varepsilon_t})$ time and proposes an* LevD *solution that contains up to $(1+\varepsilon)\mathrm{LevD}(S \times I, S')$ insertions, deletions, or substitutions with probability $1 - n^{-O_\varepsilon(\log n)}$.*

We remark that the method we use to prove Theorem 9.8.1 can be further generalized to any edit-distance-like metric that associates constant and possibly different costs for each of the insertion, deletion, and substitution operations. However, for the sake of simplicity, we only include the analysis for the Levenshtein distance.

Similar to previous sections, we assume that we have a string $S \times I$ of length n_0 consisting of a string S indexed by string I constructed as in Theorem 9.2.3. We aim to approximate the Levenshtein distance between $S \times I$ and some given string S' of length n. Note that for an $1+\varepsilon$ multiplicative approximation, one can assume, without loss of generality, that the lengths of S and S' are within ε factor of each other since, otherwise, a trivial LevD solution with $\min(|S|, |S'|)$ substitutions and $\max(|S|, |S'|) - \min(|S|, |S'|)$ would yield a $1+\varepsilon$ approximation.

In the construction of the index string I from Theorem 9.2.3, the label is constructed by appending the codewords of some constant-rate code that is L-list decodable from $1 - \varepsilon_{\mathcal{C}}$ fraction of deletions and $1 - \varepsilon_{\mathcal{C}}$ fraction of insertions with block length N where $N = O_{\varepsilon_{\mathcal{C}}}(\log n)$ and L is a sub-logarithmic function of n.

We start with an informal description of our algorithm. We split S' into intervals of length N to which we will refer as *blocks* of S'. The algorithm randomly chooses $\sqrt{n}$ blocks of S'. Running the decoder of code C on each of those blocks would give a list of L codewords or, equivalently, positions in S where that block might be mapped to the matching between S and S' that corresponds to the optimal set of insertions, deletions, and substitutions between them. At the high level, the algorithm does the following: for any pair of randomly chosen blocks that are no more than $O(\sqrt{n}\log^2 n)$ apart from each other, the algorithm decodes both blocks to find L^2 candidate substrings from S to where the substring of S' surrounded by them might be mapped. The algorithm then approximates the edit distance between the substring of S' surrounded by the pair of blocks and all L^2 substrings in S recursively and uses dynamic programming to find the optimal combination of such candidates. The algorithm is formally described in Algorithm 13.

The algorithm takes both strings, block length and the decoder of the utilized code, ε and ε_t, as well as the parameter *rec_depth* that keeps track of the depth of the recursion. The algorithm only recurs for $\left\lceil \log \frac{1}{\varepsilon_t} \right\rceil$ levels and at that level uses the quadratic time dynamic programming to compute the exact LevD on small subproblems. On earlier levels, the algorithm picks $\sqrt{n}$ blocks of S', $r_1 < r_2 < \cdots < r_{\sqrt{n}}$[2], and decodes them to

[2]We use r_i to both refer to the block itself as well as the i.d. of the block, i.e., block r_i is located at $S[(r_i - 1) \cdot N, r_i \cdot N - 1]$

find L candidate "neighborhoods" in S to which that block might be matched to in the matching that yields the optimal LevD. Then, in Algorithm 13, the algorithm loops over a neighborhood of length $O_{\varepsilon_\mathcal{C}}(N)$ around each codeword in S to "fine tune" where the last element of the randomly chosen block is matched to. For each such location in S, the algorithm aims to find the best matching between the prefix of S that ends at that location and the prefix of S' that ends at the block of S' under consideration in the dynamic programming represented by array $d[\cdot][\cdot]$.

The algorithm updates the value of each $d[i][j]$ using all $d[i'][j']$s where $i' < i$ and $j' < j$ and adding the LevD between the two added suffixes $S[i', i]$ and $S'(r_j, r_{j'})$[3]. In Algorithm 13 the algorithm checks whether the length of these suffixes are $O(\sqrt{n}\log^2 n)$ or not. If they are, the algorithm uses the recursion on Algorithm 13 to approximate the LevD between them. Otherwise, it uses the trivial conversion between the two prefixes instead. Algorithm 13 returns the final solution that is obtained by the choosing the best $d[i][j]$ plus the trivial LevD for the remaining suffixes.

9.8.1 Correctness

Parameter ε_C specifies the code $\mathcal{C}$ used in the construction of the index that can list-decode from $(1-\varepsilon_C)N$ insertions and $(1-\varepsilon_C)N$ deletions. We will do the most of the analysis relying on parameter ε_C and determine it later.

Similar to previous sections, we start by taking the matching $\mathcal{M}$ between S and S' that yields the optimal Levenshtein distance. We do the following modifications on $\mathcal{M}$.

1. If a block in S' contains less than $N\varepsilon_C$ edges, remove the edges or equivalently, in the corresponding LevD solution, convert the matches to symbol substitutions. This will increase the corresponding LevD by a multiplicative factor no larger than $(1+\varepsilon_C)$ as one can use the $N(1-\varepsilon_C)$ unmatched vertices in that block and their corresponding edits/substitutions as the budget.

2. If the projection of a block of S' in S is larger than $\frac{N}{\varepsilon}$, remove the edges or equivalently, in the corresponding LevD solution, convert the matches to symbol substitutions. This increases the size of the LevD solution up to another $(1+\varepsilon_C)$ multiplicative factor as we can use the $N \cdot \left(\frac{1}{\varepsilon} - 1\right)$ unmatched vertices in the projection of S as the approximation error budget.

Let us denote the resulting matching with $\mathcal{M}'$ and corresponding LevD solutions for $\mathcal{M}$ and $\mathcal{M}'$ with $\text{LevD}_\mathcal{M}$ and $\text{LevD}_{\mathcal{M}'}$. We know that

$$\text{LevD}_{\mathcal{M}'} \leq (1+\varepsilon_C)^2 \text{LevD}_\mathcal{M}. \tag{9.5}$$

With the following two assumptions, the LevD computed by the algorithm will be at least as good as $\text{LevD}_{\mathcal{M}'}$:

1. All recursive calls at depth 2 correctly compute the exact LevD between the input strings.

[3] $S'(r_j, r_{j'})$ represents the substring of S' that is surrounded by r_j and $r_{j'}$ including $r_{j'}$ and excluding r_j, i.e., $S'(r_j, r_{j'}) = S'[(r_j+1)\cdot N, r_{j'} \cdot N]$.

Algorithm 13 $(1+\varepsilon)$-Approximation for Levenshtein Distance in $O(n^{1+\varepsilon_t})$ Time

procedure LevD-Approx($S, S', N, \mathrm{Dec}_{\mathcal{C}}(\cdot), \varepsilon, \varepsilon_t, rec\_depth = 1$)

  if $rec\_depth = \left\lceil \log \frac{1}{\varepsilon_t} \right\rceil$ **then**

    return LevD-Exact(S, S') ▷ Compute the LevD using dynamic programming.

  $n \leftarrow |S'|$

  $\varepsilon_{\mathcal{C}} \leftarrow \frac{\varepsilon}{100 \log \frac{1}{\varepsilon_t}}$

  Randomly select $\sqrt{n}$ blocks $r_1, r_2, \cdots, r_{\sqrt{n}}$ from S' ▷ r_i's are ordered in the order of appearance in S' from left to right.

  initialize $d[\cdot][\cdot] \leftarrow \infty$ ▷ $d[i][j]$ represents the LevD between $S[1,i]$ and the substring of S' that ends at r_j. Note that for each j, we only care for up to $\frac{L^2}{\varepsilon^2}$ values for i.

  $\forall j \in [1, \sqrt{n}] : d[0][j] \leftarrow N \cdot j$

  for $j = 1$ to $\sqrt{n}$ **do**

    $l_j \leftarrow \mathrm{Dec}_{\mathcal{C}}(r_j)$

    for each $x \in l_j$ **do**

      for each i from $x \cdot N - \frac{N}{\varepsilon_{\mathcal{C}}}$ to $x \cdot N + \frac{N}{\varepsilon_{\mathcal{C}}}$ **do**

        $d[i][j] \leftarrow \max(i, r_j \cdot N)$ ▷ Trivial conversion with edit and substitutions

        for each $r_{j'}$ where $j' < j$ **do**

          for each $x' \in l_{j'}$ **do**

            for each i' from $x' \cdot N - \frac{N}{\varepsilon}$ to $x' \cdot N + \frac{N}{\varepsilon}$ where $i' < i$ **do**

              $\tilde{S} = S[i', i]$

              $\tilde{S'} = S'(r'_j, r_j)$ ▷ $S'(r'_j, r_j)$ indicates the substring of S' that start after r'_j and ends at the end of r_j

              if $|\tilde{S'}| \leq \sqrt{n} \log^2 n$ and $|\tilde{S}| \leq \frac{1}{\varepsilon_{\mathcal{C}}} \sqrt{n} \log^2 n$ **then**

                $cost \leftarrow$ LevD-Approx($\tilde{S}, \tilde{S'}, N, \mathrm{Dec}_{\mathcal{C}}(\cdot), \varepsilon, \varepsilon_t, rec\_depth + 1$)

                $d[i][j] \leftarrow \min\left(d[i][j], d[i'][j'] + cost\right)$

              else

                $d[i][j] \leftarrow \min\left(d[i][j], d[i'][j'] + \textsc{Trivial-LevD}(\tilde{S}, \tilde{S'})\right)$

                ▷ Trivial-LevD(x, y) is the size of the trivial LevD between x and y, i.e., $\max(|x|, |y|)$.

  return $\min_{j \in [1, \sqrt{n}], i} d[i][j] + \textsc{Trivial-LevD}(S[|S| - i, |S|], S'[(r_j + 1) \cdot N, |S|])$

2. Among the blocks that are chosen in the random selection and their edges are not removed in the elimination steps, no two consecutive ones are more than $\sqrt{n}\log^2 n$ apart.

Having these two assumptions, the LevD computed by the algorithm will be at least as good as $\text{LevD}_{\mathcal{M}'}$. This can be verified by considering the list of chosen blocks that fit in the second condition. For each of such blocks, the algorithm iterates over the exact location in S to where their last element is mapped in Algorithm 13. Because each two consecutive such blocks are no more than $\sqrt{n}\log^2 n$ far apart by the second condition, their corresponding sub-problems update from each other in Algorithm 13 since they satisfy the condition in Algorithm 13. Finally, as we assumed in the first condition that second depth calls all compute the exact LevD, the solution provided by $\mathcal{M}'$ appears in Algorithm 13.

In the following lemma, we show that the second condition holds with high probability, except when the density of the edges between the two selected blocks in $\mathcal{M}'$ is less than $\varepsilon_{\mathcal{C}}$. In which case, choosing the trivial matching in Algorithm 13 only adds an extra $1+\varepsilon_{\mathcal{C}}$ multiplicative factor to the approximation.

Lemma 9.8.2. *The following holds for the $\sqrt{n}$ randomly chosen blocks with probability $1 - n^{-O_{\varepsilon_{\mathcal{C}}}(\log n)}$.*

1. *In any substring of length $\sqrt{n}\log^2 n$, the number of chose blocks is in $\left[\frac{1}{100}\cdot\log^2 n, 100\cdot\log^2 n\right]$.*

2. *There is at least one selected block that is not removed in the elimination process in any substring of length $\sqrt{n}\log^2 n$ in S' whose corresponding vertices in $\mathcal{M}'$ contain more than $l\varepsilon_{\mathcal{C}}$ blocks not deleted in the elimination process.*

Therefore, with high probability, each level of recursion, adds only up to $(1+\varepsilon_{\mathcal{C}})^3$ multiplicative factor to the approximation. Hence, overall, the approximation guarantee will be $(1+\varepsilon_{\mathcal{C}})^{3\log\frac{1}{\varepsilon_t}}$. Note that with constant recursion depth, the size and the number of the sub-problems will be polynomial in terms of n. Therefore, by the Union bound and with choosing $\varepsilon_{\mathcal{C}} = \frac{\varepsilon}{100\log(1/\varepsilon_t)}$, the approximation ratio of the algorithm does not exceed $1+\varepsilon$ with high probability.

Proof of Lemma 9.8.2. To prove this lemma, we use the following probability concentration inequalities.

Lemma 9.8.3 (Chernoff Bound)**.** *Let $X_1, X_2, \cdots, X_n$ be independent random variables taking values from $\{0,1\}$ and $X = X_1 + \cdots + X_n$ and $\mu = \mathbb{E}[X]$. Then, for any $0 < \delta < 1$*

$$\Pr(X \le (1-\delta)\mu) \le e^{-\frac{\delta^2\mu}{2}}$$

and for any $\delta > 1$

$$\Pr(X \ge (1+\delta)\mu) \le e^{-\frac{\delta\mu}{3}}.$$

For a specific substring of length $\sqrt{n}\log^2 n$, each randomly chosen block lies in there with probability $\frac{\log^2 n}{\sqrt{n}}$. Let X denote the number of such blocks. Clearly, have $\mathbb{E}[X] = \log^2 n$ and, therefore, using the two forms of Chernoff inequality,

$$\Pr\left(X \notin \left[\frac{\log^2 n}{100}, 100\log^2 n\right]\right) \leq n^{-O(\log n)}.$$

Further, for the second condition, same argument with an expectation of at least $\varepsilon_{\mathcal{C}}\log^2 n$ holds. Therefore, all conditions mentioned in the statement hold with probability $1 - n^{-O_{\varepsilon_{\mathcal{C}}}(\log n)}$. □

9.8.2 Time Analysis

The algorithm picks $\sqrt{n}$ blocks and decodes each of them. Then it goes through $\sqrt{n}\cdot L\cdot\frac{1}{\varepsilon_{\mathcal{C}}}$ sub-problems. For each sub problem, the corresponding dynamic programming value is computed by looking at up to $\sqrt{n}\cdot L\cdot\frac{1}{\varepsilon_{\mathcal{C}}}$ other sub-problems and recursively calling $100\log^2 n\cdot L\cdot\frac{1}{\varepsilon_{\mathcal{C}}}$ sub-problems of size $O(\sqrt{n}\log^2 n)$.

$$\begin{aligned} T(n) &\leq \frac{L\sqrt{n}}{\varepsilon_{\mathcal{C}}}\cdot\left[T\left(\sqrt{n}\log^2 n\right) + \frac{L\sqrt{n}}{\varepsilon_{\mathcal{C}}}\right] \\ &= \frac{L^2 n}{\varepsilon_{\mathcal{C}}^2} + \frac{L\sqrt{n}}{\varepsilon_{\mathcal{C}}}T\left(\sqrt{n}\log^2 n\right) \end{aligned}$$

Since the recursion goes on for $\left\lceil\log\frac{1}{\varepsilon_t}\right\rceil$ levels, the time complexity would be as follows.

$$T(n) \leq n^{1+\varepsilon_t}\left(\frac{L\log^2 n}{\varepsilon_{\mathcal{C}}}\right)^{\log\frac{1}{\varepsilon_t}} = O_{\varepsilon_{\mathcal{C}}}\left(n^{1+\varepsilon_t}\log^{2\log\frac{1}{\varepsilon_t}} n\cdot\exp(\exp(\exp(\log^* n)))\right) = \tilde{O}_{\varepsilon_{\mathcal{C}}}(n^{1+\varepsilon_t}) \tag{9.6}$$

Chapter 10

Combinatorial Properties of Synchronization Strings

In this chapter, we study combinatorial properties of synchronization strings. In Chapter 3, we showed that for any parameter $\varepsilon > 0$, synchronization strings of arbitrary length exist over an alphabet whose size depends only on ε. Specifically, we obtain an alphabet size of $O(\varepsilon^{-4})$, which leaves an open question on where the minimal size of such alphabets lies between $\Omega(\varepsilon^{-1})$ and $O(\varepsilon^{-4})$. In this chapter, we partially bridge this gap by providing an improved lower bound of $\Omega\left(\varepsilon^{-3/2}\right)$, and an improved upper bound of $O\left(\varepsilon^{-2}\right)$. We also provide fast explicit constructions of synchronization strings over small alphabets.

Further, along the lines of previous work on similar combinatorial objects, we study the extremal question of the smallest possible alphabet size over which synchronization strings can exist for some constant $\varepsilon < 1$. We show that one can construct ε-synchronization strings over alphabets of size four while no such string exists over binary alphabets. This reduces the extremal question to whether synchronization strings exist over ternary alphabets.

10.1 Introduction

This chapter focuses on the study of the combinatorial properties of synchronization string. Informally, a synchronization string is a (finite or infinite) string that avoids similarities between pairs of intervals in the string. Such nice properties and synchronization strings themselves can actually be motivated from at least two different aspects: coding theory and pattern avoidance. Throughout previous chapter we extensively studied synchronization strings and their applications in numerous coding problems. We have also discussed how properties of synchronization strings such as the parameter ε, their alphabet size, their construction time, and their corresponding repositioning algorithms translate into the properties of the codes that are constructed using synchronization strings. To further motivate the discussions of this chapter, we now provide a brief review of the previous work on the combinatorial objects similar to synchronization strings.

10.1.1 Motivation and Previous Work in Pattern Avoidance

Apart from applications in coding theory and other communication problems involving insertions and deletions, synchronization strings are also interesting combinatorial objects from a mathematical perspective. As a matter of fact, plenty of very similar combinatorial objects have been studied prior to this work.

A classical work of Axel Thue [Thu06] introduces and studies *square-free* strings, i.e., strings that do not contain two identical consecutive substrings. Thue shows that such strings exist over alphabets of size three and provides a fast construction of such strings using *morphisms*. The seminal work of Thue inspired further works on the same problem [Thu12, Lee57, Cro82, She81, SS82, Zol15] and problems with a similar pattern avoidance theme.

Krieger et. al. [KORS07] study strings that satisfy relaxed variants of square-freeness, i.e., strings that avoid *approximate squares*. Their study provides several results on strings that avoid consecutive substrings of equal length with small additive or multiplicative Hamming distance in terms of their length. In each of these regimes, [KORS07] gives constructions of approximate square free strings over alphabets with small constant size for different parameters.

Finally, Camungol and Rampersad [CR$^+$16] study *approximate squares with respect to edit distance*, which is equivalent to the ε-synchronization string notion except that the edit distance property is only required to hold for pairs of consecutive substrings of equal length. [CR$^+$16] employs a technique based on entropy compression to prove that such strings exist over alphabets that are constant in terms of string length but exponentially large in terms of ε^{-1}. We note that in Chapter 3, we already improved this dependence to $O(\varepsilon^{-4})$.

Again, a main question addressed in most of the above-mentioned previous work on similar mathematical objects is how small the alphabet size can be.

10.1.2 Our Results

In this chapter, we study the question of how small the alphabet size of an ε-synchronization string can be. We address this question both for a specified ε and for an unspecified ε. In the first case, we try to bridge the gap between the upper bound of $O(\varepsilon^{-4})$ provided in Chapter 3 and the lower bound of $\Omega(\varepsilon^{-1})$. In the second case, we study the question of how small the alphabet size can be to ensure the existence of an ε-synchronization string for some constant $\varepsilon < 1$. In both cases, we also give efficient constructions that improve previous results.

New Bounds on Minimal Alphabet Size for a given ε

Our first theorem gives an improved upper bound and lower bound for the alphabet size of an ε-synchronization string for a given ε.

Theorem 10.1.1. *For any $0 < \varepsilon < 1$, there exists an alphabet Σ of size $O(\varepsilon^{-2})$ such that an infinite ε-synchronization string exists over Σ. In addition, $\forall n \in \mathbb{N}$, a randomized algorithm can construct an ε-synchronization string of length n in expected time $O(n^5 \log n)$. Further, the alphabet size of any ε-synchronization string that is long enough in terms of ε has to be at least $\Omega\left(\varepsilon^{-3/2}\right)$.*

Next, we provide efficient and even linear-time constructions of ε-synchronization strings over drastically smaller alphabets than the efficient constructions in Chapter 8.

Theorem 10.1.2. *For every $n \in \mathbb{N}$ and any constant $\varepsilon \in (0, 1)$, there is a deterministic construction of a (long-distance) ε-synchronization string of length n over an alphabet of size $O(\varepsilon^{-2})$ that runs in $\mathsf{poly}(n)$ time. Further, there is a highly-explicit linear time construction of such strings over an alphabet of size $O(\varepsilon^{-3})$.*

Moreover, in Section 10.4.3, we present a method to construct infinite synchronization strings using constructions for finite ones that only increases the alphabet size by a constant factor—as opposed to the construction in Chapter 8 that increases the alphabet size quadratically.

Theorem 10.1.3. *For any constant $0 < \varepsilon < 1$, there exists an explicit construction of an infinite ε-synchronization string S over an alphabet of size $O(\varepsilon^{-2})$. Further, there exists a highly-explicit construction of an infinite ε-synchronization string S over an alphabet of size $O(\varepsilon^{-3})$ such that for any $i \in \mathbb{N}$, the first i symbols can be computed in $O(i)$ time and $S[i, i + \log i]$ can be computed in $O(\log i)$ time.*

Minimal Alphabet Size for Unspecified ε: Three or Four?

One interesting question that has been commonly addressed by the previous work on similar combinatorial objects is the size of the smallest alphabet over which one can find such objects. Along the lines of [Thu06, Zol15, KORS07, CR+16], we study the existence of synchronization strings over alphabets with minimal constant size.

It is easy to observe that no such string can exist over a binary alphabet since any binary string of length four either contains two consecutive identical symbols or two consecutive identical substrings of length two. On the other hand, one can extract constructions over constant-sized alphabets from the existence proofs in Chapters 3 and 8, but the unspecified constants there would be quite large. In Section 10.6.2, for some $\varepsilon < 1$, we provide a construction of arbitrarily long ε-synchronization strings over an alphabet of size four. This narrows down the question to whether such strings exist over alphabets of size three.

To construct such strings, we introduce the notion of *weak synchronization string*, which requires substrings to satisfy a similar property as that of an ε-synchronization string, except that the lower bound on edit distance is rounded down to the nearest integer. We show that weak synchronization strings exist over binary alphabets and use one such string to modify a ternary square-free string ([Thu06]) into a synchronization string over an alphabet of size four.

Finally, in Section 10.7, we provide experimental evidence for the existence of synchronization strings over ternary alphabets by finding lower-bounds for ε for which ε-synchronization strings over alphabets of size 3, 4, 5, and 6 might exist. Similar experiments have been provided for related combinatorial objects in the previous work [KORS07, CR$^+$16].

Constructing Synchronization Strings Using Uniform Morphisms

Morphisms have been widely used in previous work as a tool to construct similar combinatorial objects. A uniform morphism of rank r over an alphabet Σ is a function $\phi : \Sigma \to \Sigma^r$ that maps any symbol of an alphabet Σ to a string of length r over the same alphabet. Using this technique, some similar combinatorial objects in previous work have been constructed by taking a symbol from the alphabet and then repeatedly using an appropriate morphism to replace each symbol with a string [Zol15, KORS07]. Here we investigate whether such tools can also be utilized to construct synchronization strings. In Section 10.6.1, we show that no such morphism can construct arbitrarily long ε-synchronization strings for any $\varepsilon < 1$.

10.2 Some Notations and Definitions

We formally introduce the *square-free* property for strings in the following.

Definition 10.2.1 (square-free string)**.** *A string S is a square free string if $\forall 1 \leq i < i + 2l \leq |S| + 1$, $(S[i, i + l)$ and $S[i + l, i + 2l))$ are different as words.*

We also introduce the following generalization of a synchronization string, which will be useful in our deterministic constructions of synchronization strings.

Definition 10.2.2 (ε-synchronization circle)**.** *A string S is an ε-synchronization circle if $\forall 1 \leq i \leq |S|$, $S_i, S_{i+1}, \ldots, S_{|}S|, S_1, S_2, \ldots, S_{i-1}$ is an ε-synchronization string.*

10.3 ε-synchronization Strings and Circles with Alphabet Size $O(\varepsilon^{-2})$

In this section, we show that by using a non-uniform sample space together with the Lovász local Lemma, we can have a randomized polynomial time construction of an ε-synchronization string with alphabet size $O(\varepsilon^{-2})$. We then use this to give a simple construction of an ε-synchronization circle with alphabet size $O(\varepsilon^{-2})$ as well. Although the constructions here are randomized, the parameter ε can be anything in $(0, 1)$ (even sub-constant), while our deterministic constructions in later sections usually require ε to be a constant in $(0, 1)$. We first recall the general Lovász local lemma.

Lemma 10.3.1. *(General Lovász Local Lemma) Let $A_1, ..., A_n$ be a set of bad events. $G(V, E)$ is a dependency graph for this set of events if $V = \{1, \ldots, n\}$ and each event A_i is mutually independent of all the events $\{A_j : (i, j) \notin E\}$. If there exists $x_1, ..., x_n \in [0, 1)$ such that for all i we have*

$$\Pr(A_i) \leq x_i \prod_{(i,j)\in E} (1 - x_j)$$

Then the probability that none of these events happens is bounded by

$$\Pr\left(\bigwedge_{i=1}^{n} \bar{A}_i\right) \geq \prod_{i=1}^{n}(1 - x_i) > 0$$

Using this lemma, we have the following theorem showing the existence of ε-synchronization strings over an alphabet of size $O(\varepsilon^{-2})$.

Theorem 10.3.2. *$\forall \varepsilon \in (0, 1)$ and $\forall n \in \mathbb{N}$, there exists an ε-synchronization string S of length n over alphabet Σ of size $\Theta(\varepsilon^{-2})$.*

Proof. Suppose $|\Sigma| = c_1\varepsilon^{-2}$ where c_1 is a constant. Let $t = c_2\varepsilon^{-2}$ and $0 < c_2 < c_1$. The sampling algorithm is as follows:

1. Randomly pick t different symbols from Σ and let them be the first t symbols of S. If $t \geq n$, we just pick n different symbols.

2. For $t + 1 \leq i \leq n$, we pick the ith symbol $S[i]$ uniformly randomly from $\Sigma \setminus \{S[i - 1], \ldots, S[i - t + 1]\}$

Now we prove that there's a positive probability that S contains no *bad* interval $S[i, k]$ which violates the requirement that $\mathrm{ED}(S[i, j], S[j+1, k]) > (1-\varepsilon)(k-i)$ for any $i < j < k$. This requirement is equivalent to $\mathrm{LCS}(S[i, j], S[j + 1, k]) < \frac{\varepsilon}{2}(k - i)$.

Notice that for $k - i \leq t$, the symbols in $S[i, k]$ are completely distinct. Hence we only need to consider the case where $k - i > t$. First, let's upper bound the probability that an

interval is bad:

$$\begin{aligned}\Pr[\text{interval I of length } l \text{ is bad}] &\leq \binom{l}{\varepsilon l}(|\Sigma|-t)^{-\frac{\varepsilon l}{2}} \\ &\leq \frac{el}{\varepsilon l}^{\varepsilon l}(|\Sigma|-t)^{-\frac{\varepsilon l}{2}} \\ &\leq \left(\frac{\varepsilon\sqrt{|\Sigma|-t}}{e}\right)^{-\varepsilon l} \\ &= C^{-\varepsilon l}\end{aligned}$$

The first inequality holds because if the interval is bad, then it has to contain a repeating sequence $a_1a_2\ldots a_pa_1a_2\ldots a_p$ where p is at least $\frac{\varepsilon l}{2}$. Such sequence can be specified via choosing εl positions in the interval and the probability that a given sequence is valid for the string in this construction is at most $(|\Sigma|-t)^{-\frac{\varepsilon l}{2}}$. The second inequality comes from Stirling's inequality.

The inequality above indicates that the probability that an interval of length l is bad can be upper bounded by $C^{-\varepsilon l}$, where C is a constant and can be arbitrarily large by modifying c_1 and c_2.

Now we use general Lovász local lemma to show that S contains no bad interval with positive probability. First we'll show the following lemma.

Claim. *The badness of interval $I = S[i,j]$ is mutually independent of the badness of all intervals that do not intersect with I.*

Proof. Suppose the intervals before I that do not intersect with I are $I_1,\ldots,I_m$, and those after I are $I'_1,\ldots,I'_{m'}$. We denote the indicator variables of each interval being bad as b, b_k and $b'_{k'}$. That is,

$$b=\begin{cases}0 & \text{if } I \text{ is not bad}\\ 1 & \text{if } I \text{ is bad}\end{cases},\quad b_k=\begin{cases}0 & \text{if } I_k \text{ is not bad}\\ 1 & \text{if } I_k \text{ is bad}\end{cases},\quad b'_{k'}=\begin{cases}0 & \text{if } I'_{k'} \text{ is not bad}\\ 1 & \text{if } I'_{k'} \text{ is bad}\end{cases}$$

First we prove that there exists $p\in(0,1)$ such that $\forall x_1,x_2,\ldots,x_m\in\{0,1\}$,

$$\Pr(b=1|b_k=x_k,k=1,\ldots,m)=p$$

According to our construction, we can see that for any fixed prefix $S[1,i-1]$, the probability that I is bad is a fixed real number p'. That is,

$$\forall \text{ valid } \tilde{S}\in\Sigma^{i-1},\Pr(b=1|S[1,i-1]=\tilde{S})=p'$$

This comes from the fact that, the sampling of the symbols in $S[i,k]$ only depends on the previous $h=\min\{i-1,t-1\}$ different symbols, and up to a relabeling these h symbols are the same h symbols (e.g., we can relabel them as $\{1,\cdots,h\}$ and the rest of the symbols as $\{h+1,\cdots,|\Sigma|\}$). On the other hand the probability that $b=1$ remains unchanged under

any relabeling of the symbols, since if two sampled symbols are the same, they will stay the same; while if they are different, they will still be different. Thus, we have:

$$\begin{aligned}
&\Pr(b=1|b_k=x_k, i=1,\ldots,m)\\
=&\frac{\Pr(b=1, b_k=x_k, i=1,\ldots,m)}{\Pr(b_k=x_k, k=1,\ldots,m)}\\
=&\frac{\sum_{\tilde{S}}\Pr(b=1, S[1,i-1]=\tilde{S})}{\sum_{\tilde{S}}\Pr(S[1,i-1]=\tilde{S})}\\
=&\sum_{\tilde{S}}\frac{\Pr(b=1, S[1,i-1]=\tilde{S})}{\Pr(S[1,i-1]=\tilde{S})}\cdot\frac{\Pr(S[1,i-1]=\tilde{S})}{\sum_{\tilde{S}'}\Pr(S[1,i-1]=\tilde{S}')}\\
=&\sum_{\tilde{S}}\Pr(b=1|S[1,i-1]=\tilde{S})\cdot\frac{\Pr(S[1,i-1]=\tilde{S})}{\sum_{\tilde{S}'}\Pr(S[1,i-1]=\tilde{S}')}\\
=&p'\sum_{\tilde{S}}\frac{\Pr(S[1,i-1]=\tilde{S})}{\sum_{\tilde{S}'}\Pr(S[1,i-1]=\tilde{S}')}\\
=&p'
\end{aligned}$$

In the equations, $\tilde{S}$ indicates all valid string that prefix $S[1,i-1]$ can be such that $b_k = x_k, k=1,\ldots,m$. Hence, b is independent of $\{b_k, k=1,\ldots,m\}$. Similarly, we can prove that the joint distribution of $\{b'_{k'}, k'=1,\ldots,m'\}$ is independent of that of $\{b, b_k, k=1,\ldots,m\}$. Hence b is independent of $\{b_k, b'_{k'}, k=1,\ldots,m, k'=1,\ldots,m'\}$, which means, the badness of interval I is mutually independent of the badness of all intervals that do not intersect with I. □

Obviously, an interval of length l intersects at most $l+l'$ intervals of length l'. To use the Lovász local lemma, we need to find a sequence of real numbers $x_{i,k} \in [0.1)$ for intervals $S[i,k]$ for which

$$\Pr(S[i,k]\text{is bad}) \le x_{i,k}\prod_{S[i,k]\cap S[i',k']\neq\emptyset}(1-x_{i',k'})$$

The rest of the proof is the same as that of Theorem 3.4.7. We propose $x_{i,k} = D^{-\varepsilon(k-i)}$ for some constant $D \ge 1$. Hence we only need to find a constant D such that for all $S[i,k]$,

$$C^{-\varepsilon(k-i)} \le D^{-\varepsilon(k-i)}\prod_{l=t}^{n}(1-D^{-\varepsilon l})^{l+(k-i)}$$

That is, for all $l' \in \{1,...,n\}$,

$$C^{-l'} \le D^{-l'}\prod_{l=t}^{n}(1-D^{-\varepsilon l})^{\frac{l+l'}{\varepsilon}}$$

which means that

$$C \ge \frac{D}{\prod_{l=t}^{n}(1-D^{-\varepsilon l})^{\frac{l/l'+1}{\varepsilon}}}$$

Notice that the righthand side is maximized when $n = \infty, l' = 1$. Hence it's sufficient to show that

$$C \geq \frac{D}{\prod_{l=t}^{\infty}(1 - D^{-\varepsilon l})^{\frac{l+1}{\varepsilon}}}$$

Let $L = \max_{D>1} \frac{D}{\prod_{l=t}^{\infty}(1 - D^{-\varepsilon l})^{\frac{l+1}{\varepsilon}}}$. We only need to guarantee that $C > L$.

We claim that $L = \Theta(1)$. Since that $t = c_2\varepsilon^{-2} = \omega\left(\frac{\log\frac{1}{\varepsilon}}{\varepsilon}\right)$,

$$\begin{aligned}
\frac{D}{\prod_{l=t}^{\infty}(1 - D^{-\varepsilon l})^{\frac{l+1}{\varepsilon}}} &< \frac{D}{\prod_{l=t}^{\infty}(1 - \frac{l+1}{\varepsilon}D^{-\varepsilon l})} && (10.1)\\
&< \frac{D}{1 - \sum_{l=t}^{\infty}\frac{l+1}{\varepsilon}D^{-\varepsilon l}} && (10.2)\\
&= \frac{D}{1 - \frac{1}{\varepsilon}\sum_{l=t}^{\infty}(l+1)D^{-\varepsilon l}}\\
&= \frac{D}{1 - \frac{1}{\varepsilon}\frac{2tD^{-\varepsilon t}}{(1-D^{-\varepsilon})^2}} && (10.3)\\
&= \frac{D}{1 - \frac{2}{\varepsilon^3}\frac{D^{-\frac{1}{\varepsilon}}}{(1-D^{-\varepsilon})^2}} && (10.4)
\end{aligned}$$

Inequality (10.1) comes from the fact that $(1-x)^{\alpha} > 1-\alpha x$, (10.2) comes from he fact that $\prod_{i=1}^{\infty}(1-x_i) \geq 1-\sum_{i=1}^{\infty}x_i$ and (10.3) is a result from $\sum_{l=t}^{\infty}(l+1)x^l = \frac{x^t(1+t-tx)}{(1-x)^2} < \frac{2tx^t}{(1-x)^2}$ which holds for $x < 1$.

We can see that for $D = 7$, $\max_{\varepsilon}\{\frac{2}{\varepsilon^3}\frac{D^{-\frac{1}{\varepsilon}}}{(1-D^{-\varepsilon})^2}\} < 0.9$. Therefore (10.4) is bounded by a constant, which means $L = \Theta(1)$ and the proof is complete. □

Using a modification of an argument from Chapter 3, we can also obtain a randomized construction.

Lemma 10.3.3. *There exists a randomized algorithm which for any $\varepsilon \in (0,1)$ and any $n \in \mathbb{N}$, constructs an ε-synchronization string of length n over alphabet of size $O(\varepsilon^{-2})$ in expected time $O(n^5 \log n)$.*

Proof. The algorithm is similar to that of Lemma 3.4.8, using the algorithmic Lovász local lemma from [MT10] and the extension in [HSS11]. It starts with a string sampled according to the sampling algorithm in the proof of Theorem 10.3.2, over alphabet Σ of size $C\varepsilon^{-2}$ for some large enough constant C. Then the algorithm checks all $O(n^2)$ intervals for a violation of the requirements for ε-synchronization string. If a bad interval is found, this interval is re-sampled by randomly choosing every symbol s.t. each one of them is different from the previous $t-1$ symbols, where $t = c'\varepsilon^{-2}$ with c' being a constant smaller than C.

One subtle point of our algorithm is the following. Note that in order to apply the algorithmic framework of [MT10] and [HSS11], one needs the probability space to be sampled from n independent random variables $\mathcal{P} = \{P_1, \cdots, P_n\}$ so that each event in the collection

$\mathcal{A} = \{A_1, \cdots, A_m\}$ is determined by some subset of $\mathcal{P}$. Then, when some bad event A_i happens, one only resamples the random variables that decide A_i. Upon first look, it may appear that in our application of the Lovász local lemma, the sampling of the i'th symbol depends on the the previous $h = \min\{i-1, t-1\}$ symbols, which again depend on previous symbols, and so on. Thus, the sampling of the i'th symbol depends on the sampling of all previous symbols. However, we can implement our sampling process as follows: for the i'th symbol we first independently generate a random variable P_i which is uniform over $\{1, 2, \cdots, |\Sigma| - h\}$, then we use the random variables $\{P_1, \cdots, P_n\}$ to decide the symbols, in the following way. Initially we fix some arbitrary order of the symbols in Σ, then for $i = 1, \cdots, n$, to get the i'th symbol, we first reorder the symbols Σ so that the previous h chosen symbols are labeled as the first h symbols in Σ, and the rest of the symbols are ordered in the current order as the last $|\Sigma| - h$ symbols. We then choose the i'th symbol as the $(h + P_i)$'th symbol in this new order. In this way, the random variables $\{P_1, \cdots, P_n\}$ are indeed independent, and the i'th symbol is indeed chosen uniformly from the $|\Sigma| - h$ symbols excluding the previous h symbols. Furthermore, the event of any interval $S[i, k]$ being bad only depends on the random variables $(P_i, \cdots, P_k)$ since no matter what the previous h symbols are, they are relabeled as $\{1, \cdots, h\}$ and the rest of the symbols are labeled as $\{h+1, \cdots, |\Sigma|\}$. From here, the same sequence of $(P_i, \cdots, P_k)$ will result in the same behavior of $S[i, k]$ in terms of which symbols are the same. We can, thus, apply the same algorithm as in Chapter 3.

Note that the time to get the i'th symbol from the random variables $\{P_1, \cdots, P_n\}$ is $O(n \log \frac{1}{\varepsilon})$ since we need $O(n)$ operations each on a symbol of size $C\varepsilon^{-2}$. Thus, resampling each interval takes $O(n^2 \log \frac{1}{\varepsilon})$ time since we need to resample at most n symbols. For every interval, the edit distance can be computed using the Wagner-Fischer dynamic programming within $O(n^2 \log \frac{1}{\varepsilon})$ time. [HSS11] shows that the expected number of re-sampling is $O(n)$. The algorithm will repeat until no bad interval can be found. Hence the overall expected running time is $O(n^5 \log \frac{1}{\varepsilon})$.

Note that without loss of generality we can assume that $\varepsilon > 1/\sqrt{n}$ because for smaller errors we can always use the indices directly, which have alphabet size n. So the overall expected running time is $O(n^5 \log n)$. □

We can now construct an ε-synchronization circle using Theorem 10.3.2.

Theorem 10.3.4. *For every $\varepsilon \in (0, 1)$ and every $n \in \mathbb{N}$, there exists an ε-synchronization circle S of length n over alphabet Σ of size $O(\varepsilon^{-2})$.*

Proof. First, by Theorem 10.3.2, we can have two ε-synchronization strings: S_1 with length $\lceil \frac{n}{2} \rceil$ over Σ_1 and S_2 with length $\lfloor \frac{n}{2} \rfloor$ over Σ_2. Let $\Sigma_1 \cap \Sigma_2 = \emptyset$ and $|\Sigma_1| = |\Sigma_2| = O(\varepsilon^{-2})$. Let S be the concatenation of S_1 and S_2. Then S is over alphabet $\Sigma = \Sigma_1 \cup \Sigma_2$ whose size is $O(\varepsilon^{-2})$. Now we prove that S is an ε-synchronization circle.

$\forall 1 \leq m \leq n$, consider string $S' = s_m, s_{m+1}, \ldots, s_n, s_1, s_2, \ldots, s_{m-1}$. Notice that for two strings T and T' over alphabet Σ, $\mathrm{LCS}(T, T') \leq \frac{\varepsilon}{2}(|T| + |T'|)$ is equivalent to $\mathrm{ED}(T, T') \geq (1 - \varepsilon)(|T| + |T'|)$. For any $i < j < k$, we call an interval $S'[i, k]$ good if $\mathrm{LCS}(S'[i, j], S'[j + 1, k]) \leq \frac{\varepsilon}{2}(k - i)$. It suffices to show that $\forall 1 \leq i, k \leq n$, the interval $S'[i, k]$ is good.

Without loss of generality let's assume $m \in [\lceil \frac{n}{2} \rceil, n]$. Intervals which are substrings of S_1 or S_2 are good intervals, since S_1 and S_2 are ε-synchronization strings. We are left with intervals crossing the ends of S_1 or S_2.

If $S'[i,k]$ contains s_n, s_1 but doesn't contain $s_{\lceil \frac{n}{2} \rceil}$: If $j < n-m+1$, then there's no common subsequence between $s'[i,j]$ and $S'[n-m+2,k]$. Thus,

$$\mathrm{LCS}(S'[i,j], S'[j+1,k]) \leq \mathrm{LCS}(S'[i,j], S'[j+1, n-m+1]) \leq \frac{\varepsilon}{2}(n-m+1-i) < \frac{\varepsilon}{2}(k-i)$$

If $j \geq n-m+1$, then there's no common subsequence between $S'[j+1,k]$ and $S'[i, n-m+1]$. Thus,

$$\mathrm{LCS}(S'[i,j], S'[j+1,k]) \leq \mathrm{LCS}(S'[n-m+2,j], S'[j+1,k]) \leq \frac{\varepsilon}{2}(k-(n-m+2)) < \frac{\varepsilon}{2}(k-i)$$

Thus, intervals of this kind are good.

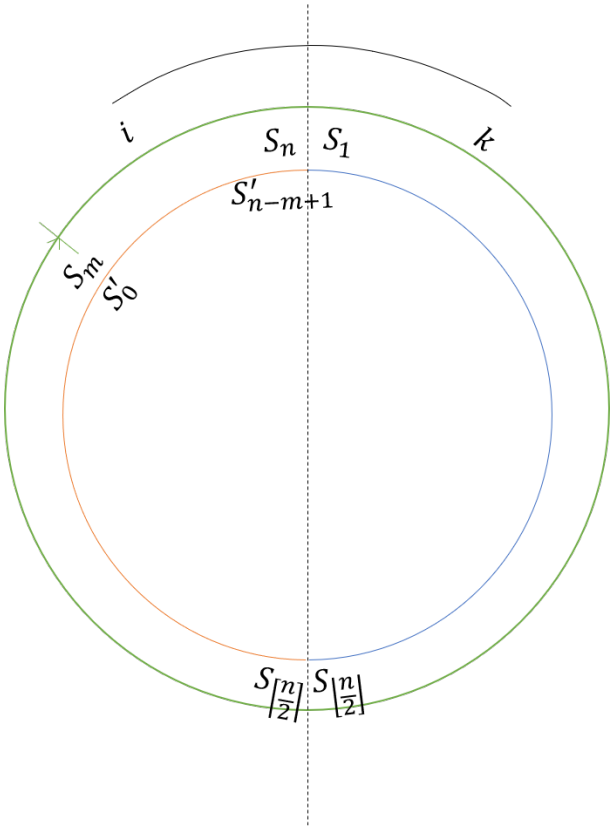


Figure 10.1: Example where $S'[i,k]$ contains s_n, s_1 but doesn't contain $s_{\lceil \frac{n}{2} \rceil}$

If $S'[i,k]$ contains $s_{\lfloor \frac{n}{2} \rfloor}, s_{\lceil \frac{n}{2} \rceil}$ but doesn't contain s_n: If $j \leq n-m+\lfloor \frac{n}{2} \rfloor + 1$, then there's no common subsequence between $S'[i,j]$ and $S'[n-m+\lceil \frac{n}{2} \rceil + 1, k]$, thus,

$$\mathrm{LCS}(S'[i,j], S'[j+1,k]) \leq \mathrm{LCS}(S'[i,j], S'[j+1, n-m+\lfloor \frac{n}{2} \rfloor + 1]) < \frac{\varepsilon}{2}(k-i)$$

If $j \geq n - m + \lfloor \frac{n}{2} \rfloor + 1$, then there's no common subsequence between $S'[j+1, k]$ and $S'[i, n - m + \lfloor \frac{n}{2} \rfloor + 1]$. Thus,

$$\text{LCS}(S'[i,j], S'[j+1,k]) \leq \text{LCS}(S'[n - m + \lceil \frac{n}{2} \rceil + 1, j], S'[j+1,k]) < \frac{\varepsilon}{2}(k-i)$$

Thus, intervals of this kind are good.

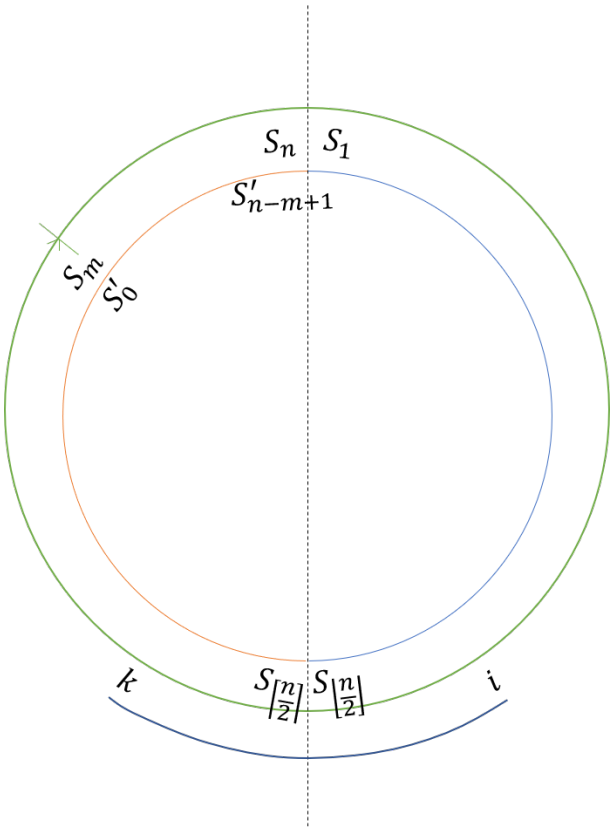


Figure 10.2: Example where $S'[i,k]$ contains $s_{\lfloor \frac{n}{2} \rfloor}, s_{\lceil \frac{n}{2} \rceil}$

If $S'[i,k]$ contains $s_{\lceil \frac{n}{2} \rceil}$ and s_n: If $n - m + 2 \leq j \leq n - m + \lfloor \frac{n}{2} \rfloor + 1$, then the common subsequence is either that of $S'[i, n-m+1]$ and $S'[n - m + \lceil \frac{n}{2} \rceil + 1, k]$ or that of $S'[n-m+2, j]$ and $S'[j+1, n - m + \lfloor \frac{n}{2} \rfloor + 1]$. This is because $\Sigma_1 \cap \Sigma_2 = \emptyset$. Thus,

$$\begin{aligned}
& \text{LCS}(S'[i,j], S'[j+1,k]) \\
\leq\ & \max\Big\{\text{LCS}(S'[i, n-m+1], S'[n - m + \lceil \frac{n}{2} \rceil + 1, k]), \\
& \qquad \text{LCS}(S'[n-m+2, j], S'[j+1, n - m + \lfloor \frac{n}{2} \rfloor + 1])\Big\} \\
<\ & \frac{\varepsilon}{2}(k-i)
\end{aligned}$$

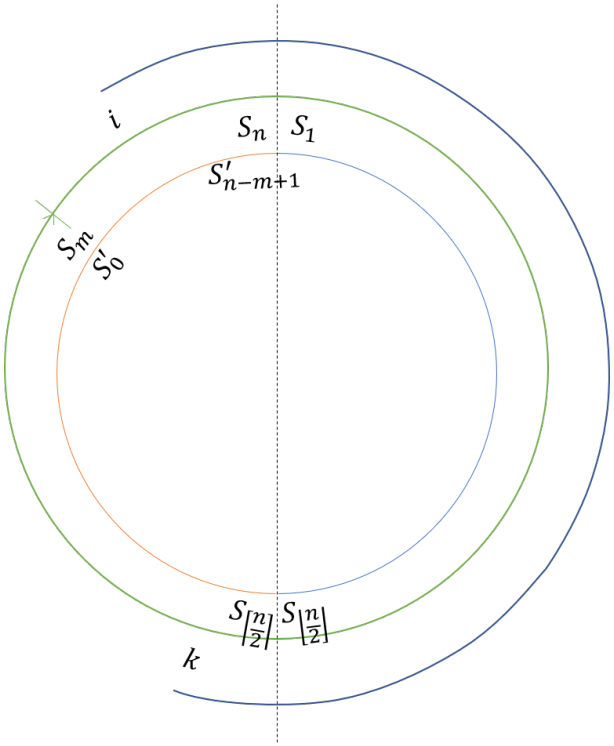


Figure 10.3: Example where $S'[i,k]$ contains $s_{\lceil \frac{n}{2} \rceil}$ and s_n

If $j \leq n-m+1$, then there's no common subsequence between $S'[i,j]$ and $S'[n-m+2, n-m+\lfloor \frac{n}{2} \rfloor + 1]$. Thus,

$$\begin{aligned}
& \mathrm{LCS}(S'[i,j], S'[j+1,k]) \\
\leq\ & \mathrm{LCS}(S'[i,j], S'[j+1,n-m+1]) + \mathrm{LCS}(S'[i,j], S'[n-m+\left\lceil \frac{n}{2} \right\rceil + 1,k]) \\
<\ & \frac{\varepsilon}{2}(n-m+1-i) + \frac{\varepsilon}{2}(n-\left\lceil \frac{n}{2} \right\rceil) \leq \frac{\varepsilon}{2}(n-m+1-i) + \frac{\varepsilon}{2}(k-(n-m+2)) \\
=\ & \frac{\varepsilon}{2}(k-1-i) < \frac{\varepsilon}{2}(k-i)
\end{aligned}$$

If $j \geq S'[n-m+\lceil \frac{n}{2} \rceil + 1]$, the proof is similar to the case where $j \leq n-m+1$.

This shows that S' is an ϵ-synchronization string. Thus, by the definition of synchronization circle, the construction gives an ϵ-synchronization circle. □

10.4 Deterministic Constructions of Long-Distance Synchronization Strings

In this section, we give deterministic constructions of synchronization strings. In fact, we consider a generalized version of synchronization strings, i.e., $f(l)$-distance ε-synchronization strings as defined in Chapter 8. Throughout this section, ε is considered to be a constant in $(0,1)$.

Definition 10.4.1 ($f(l)$-distance ε-synchronization string (Definition 8.3.1))**.** *A string $S \in \Sigma^n$ is an $f(l)$-distance ε-synchronization string if for every $1 \leq i < j \leq i' < j' \leq n+1$, $\mathrm{ED}\left(S[i,j), S[i',j')\right) > (1-\varepsilon)(l)$ for $i' - j \leq f(l)$ where $l = j + j' - i - i'$.*

As a special case, 0-distance synchronization strings are standard synchronization strings.

Similar to Chapter 8, we focus on $f(l) = n \cdot \mathbb{1}_{l > c \log n}$ where $\mathbb{1}_{l > c \log n}$ is the indicator function for $l > c \log n$. This function considers the edit distance of all pairs of large intervals and adjacent small intervals.

Definition 10.4.2 (c-long-distance ε-synchronization strings (Definition 8.3.2))**.** *We call $n \cdot \mathbb{1}_{l > c \log n}$-distance ε-synchronization strings* c-long-distance ε-synchronization strings.

10.4.1 Polynomial Time Constructions of Long-Distance Synchronization Strings

Here, by combining the deterministic Lovász local lemma of Chandrasekaran et al. [CGH13] and the non-uniform sample space used in Theorem 10.3.2, we give a deterministic polynomial-time construction of c-long ε-synchronization strings over an alphabet of size $O(\varepsilon^{-2})$. We first recall the following property of c-long synchronization strings.

Lemma 10.4.3 (Corollary 8.3.4)**.** *If S is a string which satisfies the c-long-distance ε-synchronization property for any two non-adjacent intervals of total length $2c \log n$ or less, then it satisfies the property for all pairs of non-adjacent intervals.*

We now have the following theorem.

Theorem 10.4.4. *For any $n \in \mathbb{N}$ and any constant $\varepsilon \in (0,1)$, there is a deterministic construction of a $O(1/\varepsilon)$-long-distance ε-synchronization string of length n, over an alphabet of size $O(\varepsilon^{-2})$, in time* $\mathsf{poly}(n)$.

Proof. To prove this, we will use the Lovśz local lemma and its deterministic algorithm in [CGH13]. Suppose the alphabet is Σ with $|\Sigma| = q = c_1 \varepsilon^{-2}$ where c_1 is a constant. Let $t = c_2 \varepsilon^{-2}$ and $0 < c_2 < c_1$. We denote $|\Sigma| - t$ as q. The sampling algorithm of string S (1-index based)is as follows:

- Initialize an arbitrary order for Σ.
- For ith symbol:
  - Denote $h = \min\{t-1, i-1\}$. Generate a random variable P_i uniformly over $\{1, 2, \ldots, |\Sigma| - h\}$.
  - Reorder Σ such that the previous h chosen symbols are labeled as the first h symbols in Σ, and the rest are ordered in the current order as the last $|\Sigma| - h$ symbols.
  - Choose the $(P_i + h)$'th symbol in this new order as $S[i]$.

Define the bad event A_{i_1,l_1,i_2,l_2} as intervals $S[i_1, i_1 + l_1)$ and $S[i_2, i_2 + l_2)$ violating the $c = O(1/\varepsilon)$-long-distance synchronization string property for $i_1 + l_1 \leq i_2$. In other words, A_{i_1,l_1,i_2,l_2} occurs if and only if $\mathrm{ED}(S[i_1, i_1 + l_1), S[i_2, i_2 + l_2)) \leq (1 - \varepsilon)(l_1 + l_2)$, which is equivalent to $\mathrm{LCS}(S[i_1, i_1 + l_1), S[i_2, i_2 + l_2)) \geq \frac{\varepsilon}{2}(l_1 + l_2)$.

Note that according to the definition of c-long distance ε-synchronization string and Corollary 8.3.4, we only need to consider A_{i_1,l_1,i_2,l_2} where $l_1 + l_2 < c \log n$ and $c \log n \leq l_1 + l_2 \leq 2c \log n$. Thus we can upper bound the probability of A_{i_1,l_1,i_2,l_2},

$$\begin{aligned}
\Pr\left[A_{i_1,l_1,i_2,l_2}\right] &\leq \binom{l_1}{\varepsilon(l_1+l_2)/2}\binom{l_2}{\varepsilon(l_1+l_2)/2}(|\Sigma|-t)^{-\frac{\varepsilon(l_1+l_2)}{2}} \\
&\leq \left(\frac{l_1 e}{\varepsilon(l_1+l_2)/2}\right)^{\varepsilon(l_1+l_2)/2}\left(\frac{l_2 e}{\varepsilon(l_1+l_2)/2}\right)^{\varepsilon(l_1+l_2)/2}(|\Sigma|-t)^{-\frac{\varepsilon(l_1+l_2)}{2}} \\
&= \left(\frac{2e\sqrt{l_1 l_2}}{\varepsilon(l_1+l_2)\sqrt{|\Sigma|-t}}\right)^{\varepsilon(l_1+l_2)} \\
&\leq \left(\frac{el}{\varepsilon l\sqrt{|\Sigma|-t}}\right)^{\varepsilon l} = \left(\frac{e}{\varepsilon\sqrt{|\Sigma|-t}}\right)^{\varepsilon l} = \hat{C}^{\varepsilon l},
\end{aligned}$$

where $l = l_1 + l_2$ and $\hat{C}$ is a constant which depends on c_1 and c_2.

However, to apply the deterministic Lovász local lemma (LLL), we need to have two additional requirements. The first requirement is that each bad event depends on up to logarithmically many variables, and the second is that the inequalities in the Lovśz Local Lemma hold with a constant exponential slack.

The first requirement may not be true under the current definition of badness. Consider for example the random variables $P_{i_1}, \ldots, P_{i_1+l_1-1}, P_{i_2}, P_{i_2+l_2-1}$ for a pair of split intervals $S[i_1, i_1+l_1), S[i_2, i_2+l_2)$ where the total length l_1+l_2 is at least $2c \log n$. The event A_{i_1,l_1,i_2,l_2} may depend on too many random variables (i.e., $P_{i_1}, \ldots, P_{i_2+l_2-1}$).

To overcome this, we redefine the badness of the split interval $S[i_1, i_1 + l_1)$ and $S[i_2, i_2 + l_2)$ as follows: let B_{i_1,l_1,i_2,l_2} be the event that there exists $P_{i_1+l_1}, \ldots, P_{i_2-1}$ (i.e., the random variables chosen between the two intervals) such that the two intervals generated by $P_{i_1} \ldots, P_{i_1+l_1-1}$ and $P_{i_2}, \ldots, P_{i_2+l_2-1}$ (together with $P_{i_1+l_1}, \ldots, P_{i_2-1}$) makes $\mathrm{LCS}(S[i_1, i_1+l_1), S[i_2, i_2+l_2)) \geq \frac{\varepsilon}{2}(l_1+l_2)$ according to the sampling algorithm. Note that if B_{i_1,l_1,i_2,l_2} does not happen, then certainly A_{i_1,l_1,i_2,l_2} does not happen.

Notice that with this new definition of badness, B_{i_1,l_1,i_2,l_2} is independent of $\{P_{i_1+l_1}, \ldots, P_{i_2-1}\}$ and only depends on $\{P_{i_1} \ldots, P_{i_1+l_1-1}, P_{i_2}, \ldots, P_{i_2+l_2}\}$. In particular, this implies that B_{i_1,l_1,i_2,l_2} is independent of the badness of all other intervals which have no intersection with $(S[i_1, i_1 + l_1), S[i_2, i_2 + l_2))$.

We now bound $\Pr[B_{i_1,l_1,i_2,l_2}]$. When considering the two intervals $S[i_1, i_1+l_1)$, $S[i_2, i_2+l_2)$ and their edit distance under our sampling algorithm, without loss of generality we can assume that the order of the alphabet at the point of sampling $S[i_1]$ is $(1, 2, \ldots, q)$ just by renaming the symbols. Now, if we fix the order of the alphabet at the point of sampling $S[i_2]$ in our sampling algorithm, then $S[i_2, i_2 + l_2)$ only depends on $\{P_{i_2}, \ldots, P_{i_2+l_2}\}$ and thus $\mathrm{LCS}(S[i_1, i_1 + l_1), S[i_2, i_2 + l_2))$ only depends on $\{P_{i_1} \ldots, P_{i_1+l_1-1}, P_{i_2}, \ldots, P_{i_2+l_2}\}$.

Conditioned on any fixed order of the alphabet at the point of sampling $S[i_2]$, we have that $\mathrm{LCS}(S[i_1, i_1+l_1), S[i_2, i_2+l_2)) \geq \frac{\varepsilon}{2}(l_1+l_2)$ happens with probability at most $\hat{C}^{\varepsilon l}$ by the same computation as we upper bound $\Pr[A_{i_1,l_1,i_2,l_2}]$. Note that there are at most $q!$ different orders of the alphabet. Thus, by a union bound we have

$$\Pr[B_{i_1,l_1,i_2,l_2}] \leq \hat{C}^{\varepsilon l} \times q! = C^{\varepsilon l},$$

for some constant C.

In order to meet the second requirement of the deterministic algorithm of LLL, we also need to find real numbers $x_{i_1,i_1+l_1,i_2,i_2+l_2} \in [0,1]$ such that for any B_{i_1,l_1,i_2,l_2},

$$\Pr[B_{i_1,l_1,i_2,l_2}] \leq \left[x_{i_1,l_1,i_2,l_2} \prod_{[S[i_1,i_1+l_1)\cup S[i_2,i_2+l_2)]\cap[S[i'_1,i'_1+l'_1)\cup S[i'_2,i'_2+l'_2)]\neq\emptyset} (1 - x_{i'_1,l'_1,i'_2,l'_2}) \right]^{1.01}.$$

We propose $x_{i_1,l_1,i_2,l_2} = D^{-\varepsilon(l_1+l_2)}$ for some $D > 1$ to be determined later. D has to be chosen such that for any i_1, l_1, i_2, l_2 and $l = l_1 + l_2$:

$$\left(\frac{e}{\varepsilon\sqrt{|\Sigma|}}\right)^{\varepsilon l} \leq \left[D^{-\varepsilon l} \prod_{[S[i_1,i_1+l_1)\cup S[i_2,i_2+l_2)]\cap[S[i'_1,i'_1+l'_1)\cup S[i'_2,i'_2+l'_2)]\neq\emptyset} \left(1 - D^{-\varepsilon(l'_1+l'_2)}\right) \right]^{1.01} \tag{10.5}$$

Notice that

$$\begin{aligned}
& D^{-\varepsilon l} \prod_{[S[i_1,i_1+l_1)\cup S[i_2,i_2+l_2)]\cap[S[i'_1,i'_1+l'_1)\cup S[i'_2,i'_2+l'_2)]\neq\emptyset} \left(1 - D^{-\varepsilon(l'_1+l'_2)}\right) && (10.6)\\
\geq\ & D^{-\varepsilon l} \prod_{l'=c\log n}^{2c\log n} \prod_{l'_1=1}^{l'} \left(1 - D^{-\varepsilon l'}\right)^{\left[(l_1+l'_1)+(l_1+l'_2)+(l_2+l'_1)+(l_2+l'_2)\right]n} \\
& \times \prod_{l''=t}^{c\log n} \left(1 - D^{-\varepsilon l''}\right)^{l+l''} && (10.7)\\
=\ & D^{-\varepsilon l} \prod_{l'=c\log n}^{2c\log n} \prod_{l'_1=1}^{l'} \left(1 - D^{-\varepsilon l'}\right)^{4(l+l')n} \times \prod_{l''=t}^{c\log n} \left(1 - D^{-\varepsilon l''}\right)^{l+l''} && (10.8)\\
=\ & D^{-\varepsilon l} \prod_{l'=c\log n}^{2c\log n} \left(1 - D^{-\varepsilon l'}\right)^{4l'(l+l')n} \times \left[\prod_{l''=t}^{c\log n} \left(1 - D^{-\varepsilon l''}\right)\right]^{l} \times \prod_{l''=t}^{c\log n} \left(1 - D^{-\varepsilon l''}\right)^{l''} && (10.9)\\
\geq\ & D^{-\varepsilon l} \left(1 - \sum_{l'=c\log n}^{2c\log n} (4l'(l+l')n)\, D^{-\varepsilon l'}\right) \\
& \times \left[1 - \sum_{l''=t}^{c\log n} D^{-\varepsilon l''}\right]^{l} \times \left(1 - \sum_{l''=t}^{c\log n} l'' D^{-\varepsilon l''}\right) && (10.10)
\end{aligned}$$

$$
\begin{aligned}
&\geq D^{-\varepsilon l}\left(1-\sum_{l'=c\log n}^{2c\log n}\left(4\cdot 2c\log n(2c\log n+2c\log n)n\right)D^{-\varepsilon l'}\right) && (10.11)\\
&\quad\times\left[1-\sum_{l''=t}^{\infty}D^{-\varepsilon l''}\right]^{l}\times\left(1-\sum_{l''=t}^{\infty}l''D^{-\varepsilon l''}\right) && (10.12)\\
&= D^{-\varepsilon l}\left(1-\sum_{l'=c\log n}^{2c\log n}\left(32c^2n\log^2 n\right)D^{-\varepsilon l'}\right)\times\left[1-\frac{D^{-c_2\varepsilon^{-1}}}{1-D^{-\varepsilon}}\right]^{l}\\
&\quad\times\left(1-\frac{D^{-c_2/\varepsilon}(D^{-\varepsilon}+c_2/\varepsilon^2-c_2D^{-\varepsilon}/\varepsilon^2)}{(1-D^{-\varepsilon})^2}\right) && (10.13)\\
&\geq D^{-\varepsilon l}\left(1-32c^3n\log^3 nD^{-\varepsilon c\log n}\right)\left[1-\frac{D^{-c_2\varepsilon^{-1}}}{1-D^{-\varepsilon}}\right]^{l}\\
&\quad\times\left(1-\frac{D^{-c_2/\varepsilon}(D^{-\varepsilon}+c_2/\varepsilon^2-c_2D^{-\varepsilon}/\varepsilon^2)}{(1-D^{-\varepsilon})^2}\right) && (10.14)
\end{aligned}
$$

(10.7) holds because there are two kinds of pairs of intervals. The first kind contains all pairs of intervals whose total length is between $c\log n$ and $2c\log n$ intersecting with $S[i_1,i_1+l_1)$ or $S[i_2,i_2+l_2)$. The number of such pairs is at most $(l_1+l_1')+(l_1+l_2')+(l_2+l_1')+(l_2+l_2')$. The second kind contains all adjacent intervals of total length less than $c\log n$. Notice that according to our sampling algorithm, every t consecutive symbols are distinct, thus any adjacent intervals whose total length is less than t cannot be bad. Hence the second term contains intervals such that $t\leq l''=l_1''+l_2''\leq c\log n$. The rest of the proof is the same as that Theorem 8.3.5.

(10.10) comes from the fact that for $0<x,y<1$:

$$(1-x)(1-y)>1-x-y$$

For $D=2$ and $c=2/\varepsilon$,

$$\lim_{\varepsilon\to 0}\frac{2^{-c_2/\varepsilon}}{1-2^{-\varepsilon}}=0$$

Thus, for sufficiently small ε, $\frac{2^{-c_2/\varepsilon}}{1-2^{-\varepsilon}}<\frac{1}{2}$. Moreover,

$$32c^2n\log^2 nD^{-\varepsilon l'}=\frac{2^8}{\varepsilon^3}\frac{\log^3 n}{n}=o(1)$$

Finally, for sufficiently small ε, $1-\frac{D^{-c_2/\varepsilon}(D^{-\varepsilon}+c_2/\varepsilon^2-c_2D^{-\varepsilon}/\varepsilon^2)}{(1-D^{-\varepsilon})^2}>2^{-\varepsilon}$. Therefore, for sufficiently small ε and sufficiently large n, (10.14) is satisfied under the condition:

$$
\begin{aligned}
&D^{-\varepsilon l}\prod_{[S[i_1,i_1+l_1)\cup S[i_2,i_2+l_2)]\cap[S[i_1',i_1'+l_1')\cup S[i_2',i_2'+l_2')]\neq\emptyset}\left(1-D^{-\varepsilon(l_1'+l_2')}\right)\\
&\geq 2^{-\varepsilon l}\left(1-\frac{1}{2}\right)(2^{-\varepsilon})^{l}\left(1-\frac{1}{2}\right)\geq\frac{4^{-\varepsilon l}}{4}
\end{aligned}
$$

So for LLL to work, the following should be guaranteed:

$$\left(\frac{e}{\varepsilon\sqrt{|\Sigma|-t}}\right)^{\frac{\varepsilon l}{1.01}} \leq \frac{4^{-\varepsilon l}}{4} \Leftarrow \frac{4^{2.02(1+\varepsilon)e^2}}{\varepsilon^2} \leq |\Sigma| - t$$

Hence the second requirement holds for $|\Sigma| - t = \frac{4^{4.04}e^2}{\varepsilon^2} = O(\varepsilon^{-2})$. □

Corollary 10.4.5. *For any $n \in \mathbb{N}$ and any constant $\varepsilon \in (0,1)$, there is a deterministic construction of an ε-synchronization string of length n, over an alphabet of size $O(\varepsilon^{-2})$, in time* $\mathsf{poly}(n)$.

By a similar concatenation construction used in the proof of Theorem 10.3.4, we also have a deterministic construction for synchronization circles.

Corollary 10.4.6. *For any $n \in \mathbb{N}$ and any constant $\varepsilon \in (0,1)$, there is a deterministic construction of an ε-synchronization circle of length n, over an alphabet of size $O(\varepsilon^{-2})$, in time* $\mathsf{poly}(n)$.

10.4.2 Deterministic linear time constructions of c-long distance ε-synchronization string

Here we give a much more efficient construction of a c-long distance ε-synchronization string, using synchronization circles and standard error correcting codes. We show that the following algorithm gives a construction of c-long distance synchronization strings.

Algorithm 14 Explicit Linear Time Construction of c-long distance ε-synchronization string

Input:

- An ECC $\hat{\mathcal{C}} \subset \Sigma_{\hat{C}}^m$, with distance δm and block length $m = c \log n$.
- An ε_0-synchronization circle $SC = (sc_1, \ldots, sc_m)$ of length m over alphabet Σ_{SC}.

Operations:

- Construct a code $\mathcal{C} \subset \Sigma^m$ such that

$$\mathcal{C} = \{((\hat{c}_1, sc_1), \ldots, (\hat{c}_m, sc_m))|(\hat{c}_1, \ldots, \hat{c}_m) \in \hat{\mathcal{C}}\}$$

  where $\Sigma = \Sigma_{\hat{C}} \times \Sigma_{SC}$.
- Let S be concatenation of all codewords $\mathcal{C}_1, \ldots, \mathcal{C}_N$ from $\mathcal{C}$.

Output: S.

To prove the correctness, we first recall the following theorem from Chapter 3.

Theorem 10.4.7 (Implied by Theorems 3.3.2 and 3.5.14)**.** *Given an ε_0-synchronization string S with length n, and an efficient ECC $\mathcal{C}$ with block length n, that corrects up to $n\delta\frac{1+\varepsilon_0}{1-\varepsilon_0}$ half-errors, one can obtain an insertion/deletion code $\mathcal{C}'$ that can be decoded from up to $n\delta$ deletions, where $\mathcal{C}' = \{(c'_1, \ldots, c'_n)|\forall i \in [n], c'_i = (c_i, S[i]), (c_1, \ldots, c_n) \in \mathcal{C}\}$.*

We have the following property of longest common subsequence.

Lemma 10.4.8. *Suppose T_1 is the concatenation of ℓ_1 strings, $T_1 = S_1 \circ \cdots \circ S_{\ell_1}$ and T_2 is the concatenation of ℓ_2 strings, $T_2 = S'_1 \circ \cdots \circ S'_{\ell_2}$. If there exists an integer t such that for all i, j, we have $\mathrm{LCS}(S_i, S'_j) \leq t$, then we have $\mathrm{LCS}(T_1, T_2) \leq (\ell_1 + \ell_2)t$.*

Proof. We rename the strings in T_2 by $S_{\ell_1+1}, \cdots, S_{\ell_1+\ell_2}$. Suppose the longest common subsequence between T_1 and T_2 is $\tilde{T}$, which can be viewed as a matching between T_1 and T_2. We can divide $\tilde{T}$ sequentially into disjoint intervals, where each interval corresponds to a common subsequence between a different pair of strings (S_i, S_j), where S_i is from T_1 and S_j is from T_2. In addition, if we look at the intervals from left to right, then for any two consecutive intervals and their corresponding pairs (S_i, S_j) and $(S_{i'}, S_{j'})$, we must have $i' \geq i$ and $j' \geq j$ since the matchings which correspond to two intervals cannot cross each other. Furthermore either $i' > i$ or $j' > j$ as the pair (S_i, S_j) is different from $(S_{i'}, S_{j'})$.

Thus, starting from the first interval, we can label each interval with either i or j such that every interval receives a different label, as follows. We label the first interval using either i or j. Then, assuming we have already labeled some intervals and now look at the next interval. Without loss of generality assume that the previous interval is labeled using i, now if the current $i' > i$ then we can label the current interval using i'; otherwise we must have $j' > j$ so we can label the current interval using j'. Thus the total number of the labels is at most $l_1 + l_2$, which means the total number of the intervals is also at most $l_1 + l_2$. Note that each interval has length at most t, therefore we can upper bound $\mathrm{LCS}(T_1, T_2)$ by $(l_1 + l_2)t$. □

Lemma 10.4.9. *The output S in Algorithm 14 is an ε_1-synchronization circle, where $\varepsilon_1 \leq 10(1 - \frac{1-\varepsilon_0}{1+\varepsilon_0}\delta)$.*

Proof. Suppose $\hat{\mathcal{C}}$ can correct up to δm half-errors. Then according to lemma 10.4.7, $\mathcal{C}$ can correct up to $\frac{1-\varepsilon_0}{1+\varepsilon_0}\delta m$ deletions.

Let $\alpha = 1 - \frac{1-\varepsilon_0}{1+\varepsilon_0}\delta$. Notice that $\mathcal{C}$ has the following properties:

1. $\mathrm{LCS}(\mathcal{C}) = \max_{c_1, c_2 \in C} \mathrm{LCS}(c_1, c_2) \leq \alpha m$

2. Each codeword in $\mathcal{C}$ is an ε-synchronization circle over Σ.

Consider any shift of the start point of S, we only need to prove that $\forall 1 \leq i < j < k \leq n, \mathrm{LCS}(S[i, j], S[j + 1, k]) < \frac{\varepsilon_1}{2}(k - i)$.

Suppose $S_1 = S[i, j]$ and $S_2 = S[j + 1, k]$. Let $\varepsilon_1 = 10\alpha$.

Case 1: $k - i > m$. Let $|S_1| = s_1$ and $|S_2| = s_2$, thus $s_1 + s_2 > m$. If we look at each S_h for $h = 1, 2$, then S_h can be divided into some consecutive codewords, plus at most two incomplete codewords at both ends. In this sense each S_h is the concatenation of ℓ_h strings with $\ell_h < \frac{s_h}{m} + 2$. An example of the worst case appears in Figure 10.4.

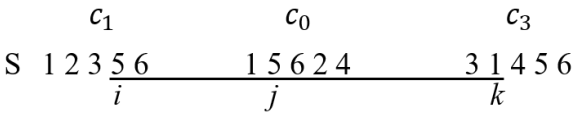


Figure 10.4: Example of the worst case, where j splits a codeword, and there are two incomplete codewords at both ends.

Now consider the longest common subsequence between any pair of these strings where one is from S_1 and the other is from S_2, we claim that the length of any such longest common subsequence is at most αm. Indeed, if the pair of strings are from two different codewords, then by the property of the code $\mathcal{C}$ we know the length is at most αm. On the other hand, if the pair of strings are from a single codeword (this happens when j splits a codeword, or when $S[i]$ and $S[k]$ are in the same codeword), then they must be two disjoint intervals within a codeword. In this case, by the property that any codeword is also a synchronization circle, the length of the longest common subsequence of this pair is at most $\frac{\varepsilon_0}{2}m$.

Note that $\alpha = 1 - \frac{1-\varepsilon_0}{1+\varepsilon_0}\delta \geq 1 - \frac{1-\varepsilon_0}{1+\varepsilon_0} = \frac{2\varepsilon_0}{1+\varepsilon_0} \geq \varepsilon_0$ (since $\delta, \varepsilon_0 \in (0,1)$). Thus $\frac{\varepsilon_0}{2}m < \alpha m$. Therefore, by Lemma 10.4.8, we have

$$\mathrm{LCS}(S_1, S_2) < \left(\frac{s_1}{m} + 2 + \frac{s_2}{m} + 2\right)\alpha m = \alpha(s_1{+}s_2{+}4m) < 5\alpha(s_1{+}s_2) = 5\alpha(k{-}i) = \frac{\varepsilon_1}{2}(k{-}i)$$

Case 2: If $k - i \leq m$, then according to the property of synchronization circle SC, we know that the longest common subsequence of S_1 and S_2 is less than $\frac{\varepsilon_0}{2}(k-i) \leq \alpha(k-i) \leq \frac{\varepsilon_1}{2}(k-i)$.

As a result, the longest common subsequence of $S[i,j]$ and $S[j+1,k]$ is less than $\frac{\varepsilon_1}{2}(k-i)$, which means that S is an ε_1-synchronization circle. □

Similarly, we also have the following lemma.

Lemma 10.4.10. *The output S of algorithm 14 is a c-long distance ε-synchronization string of length $n = Nm$ where N is the number of codewords in $\mathcal{C}$, $\varepsilon = 12(1 - \frac{1-\varepsilon_0}{1+\varepsilon_0}\delta)$.*

Proof. By Lemma 10.4.9, S is an ε_1-synchronization string, thus the length of longest common subsequence for adjacent intervals S_1, S_2 with total length $l < c\log n$ is less than $\frac{\varepsilon_1}{2}l$. We only need to consider pair of intervals S_1, S_2 whose total length $l \in [c\log n, 2c\log n]$.

Notice that the total length of S_1 and S_2 is at most $2c\log n$, which means that S_1 and S_2 each intersects with at most 3 codewords from $\mathcal{C}$. Using Lemma 10.4.8, we have that $\mathrm{LCS}(S_1, S_2) \leq 6\alpha l$. Thus, picking $\varepsilon = \max\{12\alpha, \varepsilon_1\} = 12\alpha = 12(1 - \frac{1-\varepsilon_0}{1+\varepsilon_0}\delta)$, S from Algorithm 14 is a c-long distance ε-synchronization circle. □

We need the following code constructed by Guruswarmi and Indyk [GI05].

Lemma 10.4.11 (Theorem 3 of [GI05])**.** *For every $0 < r < 1$, and all sufficiently small $\varepsilon > 0$, there exists a family of codes of rate r and relative distance $(1 - r - \varepsilon)$ over an alphabet of size $2^{O(\varepsilon^{-4}r^{-1}\log(1/\varepsilon))}$ such that codes from the family can be encoded in linear time and can also be uniquely decoded in linear time from $2(1 - r - \varepsilon)$ fraction of half errors.*

Lemma 10.4.12 (ECC by Brute Force Search)**.** *For any $n \in \mathbb{N}$, any $\varepsilon \in [0,1]$, one can construct a ECC in time $O(2^{\varepsilon n}(\frac{2e}{\varepsilon})^n n \log(1/\varepsilon))$ and space $O(2^{\varepsilon n} n \log(1/\varepsilon))$, with block length n, number of codewords $2^{\varepsilon n}$, distance $d = (1-\varepsilon)n$, alphabet size $2e/\varepsilon$.*

We can now use Algorithm 14 to give a linear time construction of $c - long - distance$ ε-synchronization strings.

Theorem 10.4.13. *For every $n \in \mathbb{N}$ and any constant $0 < \varepsilon < 1$, there is a deterministic construction of a $c = O(\varepsilon^{-2})$-long-distance ε-synchronization string $S \in \Sigma^n$ where $|\Sigma| = O(\varepsilon^{-3})$, in time $O(n)$. Moreover, $S[i, i + \log n]$ can be computed in $O(\frac{\log n}{\varepsilon^2})$ time.*

Proof. Suppose we have an error correcting code $\hat{\mathcal{C}}$ with distance rate $1-\varepsilon' = \frac{1+\frac{\varepsilon}{36}}{1-\frac{\varepsilon}{36}}(1-\frac{\varepsilon}{12})$, message rate $r_c = O(\varepsilon'^2)$, over an alphabet of size $|\Sigma_c| = O(\varepsilon'^{-1})$, with block length $m = O(\varepsilon'^{-2}\log n)$. Let $c = O(\varepsilon'^{-2}) = O(\varepsilon^{-2})$. We apply Algorithm 14, using $\hat{C}$ and an $\frac{\varepsilon}{36}$-synchronization circle SC of length m over an alphabet of size $O(\varepsilon^{-2})$. Here SC is constructed by Corollary 10.4.6 in time $\mathsf{poly}(m) = \mathsf{poly}(\log n)$. By Lemma 10.4.10, we have a c-long-distance $12(1 - \frac{1-\frac{\varepsilon}{36}}{1+\frac{\varepsilon}{36}}(1-\varepsilon')) = \varepsilon$-synchronization string of length $m \cdot |\Sigma_c|^{r_c m} \geq n$.

It remains to show that we can have such a $\hat{\mathcal{C}}$ with linear time encoding. We use the code in Lemma 10.4.11 as the outer code and the one in Lemma 10.4.12 as inner code. Let $\mathcal{C}_{out}$ be an instantiation of the code in Lemma 10.4.11 with rate $r_o = \varepsilon_o = \frac{1}{3}\varepsilon'$, relative distance $d_o = (1 - 2\varepsilon_o)$ and alphabet size $2^{O(\varepsilon_o^{-5}\log(1/\varepsilon_o))}$, and block length $n_o = \frac{\varepsilon_o^4 \log n}{\log(1/\varepsilon_o)}$, which is encodable and decodable in linear time.

Further, according to Lemma 10.4.12 one can find a code $\mathcal{C}_{in}$ with rate $r_i = O(\varepsilon_i)$ where $\varepsilon_i = \frac{1}{3}\varepsilon'$, relative distance $1 - \varepsilon_i$, over an alphabet of size $\frac{2e}{\varepsilon_i}$, and block length $n_i = O(\varepsilon_i^{-6}\log(1/\varepsilon_i))$. Note that since the block length and alphabet size are both constant because ε is a constant. So the encoding can be done in constant time.

Concatenating $\mathcal{C}_{out}$ and $\mathcal{C}_{in}$ gives the desire code $\hat{\mathcal{C}}$ with rate $O(\varepsilon'^2)$, distance $1 - O(\varepsilon')$ and alphabet of size $O(\varepsilon'^{-1})$ and block length $O(\varepsilon'^{-2}\log n)$. Moreover, the encoding of $\hat{\mathcal{C}}$ can be done in linear time, because the encoding of $\mathcal{C}_{out}$ is in linear time and the encoding of $\mathcal{C}_{in}$ is in constant time.

Note that since every codeword of $\hat{\mathcal{C}}$ can be computed in time $O(\frac{\log n}{\varepsilon^2})$, $S[i, i + \log n]$ can be computed in $O(\frac{\log n}{\varepsilon^2})$ time. □

Corollary 10.4.14. *For every $n \in \mathbb{N}$ and any constant $0 < \varepsilon < 1$, there is a deterministic construction of an ε-synchronization string $S \in \Sigma^n$ where $|\Sigma| = O(\varepsilon^{-3})$, in time $O(n)$. Moreover, $S[i, i + \log n]$ can be computed in $O(\frac{\log n}{\varepsilon^2})$ time.*

10.4.3 Explicit Constructions of Infinite Synchronization Strings

In this section, we give construction algorithms of infinite synchronization strings. To measure the efficiency of the construction of an infinite string, we consider the time complexity for computing the first n elements of that string. Remember from Chapter 8 that an infinite synchronization string is said to have an explicit construction if there is an algorithm that

computes any position $S[i]$ in time $\mathsf{poly}(i)$. Moreover, it is said to have a highly-explicit construction if there is an algorithm that computes any position $S[i]$ in time $O(\log i)$.

We have the following algorithm.

Algorithm 15 Construction of infinite ε-synchronization string

Input:

- A constant $\varepsilon \in (0, 1)$.

Operations:

- Let $q \in \mathbb{N}$ be the size of an alphabet large enough to construct an $\frac{\varepsilon}{2}$-synchronization string. Let Σ_1 and Σ_2 be two alphabets of size q such that $\Sigma_1 \cap \Sigma_2 = \emptyset$.
- Let $k = \frac{4}{\varepsilon}$. For $i = 1, 2, \ldots$, construct an $\frac{\varepsilon}{2}$-synchronization string S_{k^i} of length k^i, where S_{k^i} is over Σ_1 if i is odd and over Σ_2 otherwise.
- Let S be the sequential concatenation of $S_k, S_{k^2}, S_{k^3}, \ldots, S_{k^t}, \ldots$

Output: S.

Lemma 10.4.15. *If there is a construction of $\frac{\varepsilon}{2}$-synchronization strings with alphabet size q, then Algorithm 15 constructs an infinite ε-synchronization string with alphabet size $2q$.*

Proof. Fig. 10.5 depicts the construction proposed by Algorithm 15.

Figure 10.5: S_k and S_{k^3} are over alphabet Σ_1 and S_{k^2} is over Σ_2.

Now we show that S is an infinite ε-synchronization string.

Claim. *Let $x < y < z$ be positive integers and let t be such that $k^t \leq |S[x, z)| < k^{t+1}$. Then* $\mathrm{ED}(S[x, y), S[y, z)) \geq (1 - \frac{\varepsilon}{2})(z - x)(1 - \frac{2}{k})$.

Proof. Let l_i be the index of S where $S_{k^{i+1}}$ starts. Then $l_i = \sum_{j=1}^{i} k^j = \frac{k^{i+1}-k}{k-1}$. Notice that $l_{t-1} < 2k^{t-1}$ and $|S[x, z)| \geq k^t$, one can throw away all elements of $S[x, z)$ whose indices are less than l_{t-1} without losing more than $\frac{2k^{t-1}}{k^t} = \frac{2}{k}$ fraction of the elements of $S[x, z)$. We use $S[x', z)$ to denote the substring after throwing away the symbols before l_{t-1}. Thus $x' \geq l_{t-1}$.

Since $x' \geq l_{t-1}$, $S[x', z)$ either entirely falls into a synchronization string S_{k^l} or crosses two synchronization strings S_{k^l} and $S_{k^{l+1}}$ over two entirely different alphabets Σ_1 and Σ_2. Thus the edit distance of $S[x', y)$ and $S[y, z)$ is at least $(1 - \frac{\varepsilon}{2})(z - x)$. □

Since $k = \frac{4}{\varepsilon}$, we have that

$$\mathrm{ED}(S[x, y), S[y, z)) \geq \left(1 - \frac{\varepsilon}{2}\right)(z - x)\left(1 - \frac{2}{k}\right) = \left(1 - \frac{\varepsilon}{2}\right)^2 (z - x) \geq (1 - \varepsilon)(z - x).$$

This shows that S is an ε-synchronization string. □

If we instantiate Algorithm 15 using Corollary 10.4.5, then we have the following theorem.

Theorem 10.4.16. *For any constant $0 < \varepsilon < 1$, there exists an explicit construction of an infinite ε-synchronization string S over an alphabet of size $O(\varepsilon^{-2})$.*

Proof. We combine Algorithm 15 and Corollary 10.4.5. In the algorithm, we can construct every substring S_{k^i} in polynomial time with alphabet size $q = O(\varepsilon^{-2})$, by Corollary 10.4.5. So the first n symbols of S can be computed in polynomial time.

By Lemma 10.4.15, S is an infinite ε-synchronization string over an alphabet of size $2q = O(\varepsilon^{-2})$. □

If we instantiate Algorithm 15 using Corollary 10.4.14, then we have the following theorem.

Theorem 10.4.17. *For any constant $0 < \varepsilon < 1$, there exists a highly-explicit construction of an infinite ε-synchronization string S over an alphabet of size $O(\varepsilon^{-3})$. Moreover, for any $i \in \mathbb{N}$, the first i symbols can be computed in $O(i)$ time and $S[i, i + \log i]$ can be computed in $O(\log i)$ time.*

Proof. Combine Algorithm 15 and Corollary 10.4.14. In the algorithm, we can construct every substring S_{k^i} in linear time with alphabet size $q = O(\varepsilon^{-3})$, by Corollary 10.4.14. So the first i symbols can be computed in $O(i)$ time. Also any substring $S[i, i + \log i]$ can be computed in time $O(\log i)$.

By Lemma 10.4.15, S is an infinite ε-synchronization string over an alphabet of size $2q = O(\varepsilon^{-3})$. □

10.5 $\Omega\left(\varepsilon^{-3/2}\right)$ Lower-Bound on Alphabet Size

The *twin word* problem was introduced by Axenovich, Person, and Puzynina [APP13] and further studied by Bukh and Zhou [BZ16]. Any set of two identical disjoint subsequences in a given string is called a twin word. [APP13, BZ16] provided a variety of results on the relations between the length of a string, the size of the alphabet over which it is defined, and the size of the longest twin word it contains. We will make use of the following result from [BZ16] that is built upon Lemma 5.9 from [BHN08] to provide a new lower-bound on the alphabet size of synchronization strings.

Theorem 10.5.1 (Theorem 3 from [BZ16])**.** *There exists a constant c so that every word of length n over a q-letter alphabet contains two disjoint equal subsequences of length $cnq^{-2/3}$.*

Further, Theorem 3.5.4 states that any ε-synchronization string of length n has to satisfy *ε-self-matching property* which essentially means that it cannot contain two (not necessarily disjoint) subsequences of length εn or more. These two requirements lead to the following inequality for an ε-synchronization string of length n over an alphabet of size q.

$$cnq^{-2/3} \leq \varepsilon n \Rightarrow c'\varepsilon^{-3/2} \leq q$$

10.6 Synchronization Strings over Small Alphabets

In this section, we focus on synchronization strings over small constant-sized alphabets. We study the question of what is the smallest possible alphabet size over which arbitrarily long ε-synchronization strings can exist for some $\varepsilon < 1$, and how such synchronization strings can be constructed.

Throughout this section, we will make use of square-free strings introduced by Thue [Thu06], which is a weaker notion than synchronization strings that requires all consecutive equal-length substrings to be non-identical. Note that no synchronization strings or square-free strings of length four or more exist over a binary alphabet since a binary string of length four either contains two consecutive similar symbols or two identical consecutive substrings of length two. However, for ternary alphabets, arbitrarily long square-free strings exist and can be constructed efficiently using *uniform morphism* [Zol15]. In Section 10.6.1, we will briefly review this construction and show that no uniform morphism can be used to construct arbitrary long synchronization strings. In Section 10.6.2, we make use of ternary square-free strings to show that arbitrarily long ε-synchronization strings exist over alphabets of size four for some $\varepsilon < 1$. Finally, in Section 10.7, we provide experimental lower-bounds on ε' for which ε-synchronization strings exist over alphabets of size 3, 4, 5, and 6.

10.6.1 Morphisms cannot Generate Synchronization Strings

Previous works show that one can construct infinitely long square-free or approximate-square-free strings using *uniform morphisms*. A uniform morphism of rank r over an alphabet Σ is a function $\phi : \Sigma \to \Sigma^r$ that maps any symbol out of an alphabet Σ to a string of length r over the same alphabet. Applying the function ϕ over some string $S \in \Sigma^*$ is defined as replacing each symbol of S with $\phi(S)$.

[KORS07, Thu12, Lee57, Cro82, Zol15] show that there are uniform morphisms that generate the combinatorial objects they study respectively. More specifically, one can start from any letter of the alphabet and repeatedly apply the morphism on it to construct those objects. For instance, using the uniform morphisms of rank 11 suggested in [Zol15], all such strings will be square-free. In this section, we investigate the possibility of finding similar constructions for synchronization strings. We will show that no such morphism can possibly generate an infinite ε-synchronization strings for any fixed $0 < \varepsilon < 1$.

The key to this claim is that a matching between two substrings is preserved under an application of the uniform morphism ϕ. Hence, we can always increase the size of a matching between two substrings by applying the morphism sufficiently many times, and then adding new matches to the matching from previous steps.

Theorem 10.6.1. *Let ϕ be a uniform morphism of rank r over alphabet Σ. Then ϕ does not generate an infinite ε-synchronization string, for any $0 < \varepsilon < 1$.*

Proof. To prove this, we show that for any $0 < \varepsilon < 1$, applying morphism ϕ sufficiently many times over any symbol of alphabet Σ produces a strings that has two neighboring intervals which contradict ε-synchronization property. First, we claim that, without loss

of generality, it suffices to prove this for morphisms ϕ for which $\phi(\sigma)$ contains all elements of Σ for any $\sigma \in \Sigma$. To see this, consider the graph G with $|\Sigma|$ vertices where each vertex corresponds to a letter of the alphabet and there is a (σ_1, σ_2) edge if $\phi(\sigma_1)$ contains σ_2. It is straightforward to verify that after applying morphism ϕ over a letter sufficiently many times, the resulting string can be split into a number of substrings so that the symbols in any of them belong to a subset of Σ that corresponds to some strongly connected component in G. As ε-synchronization string property is a hereditary property over substrings, this gives that one can, without loss of generality, prove the above-mentioned claim for morphisms ϕ for which the corresponding graph G is strongly connected. Further, let d be the greatest common divisor of the size of all cycles in G. One can verify that, for some sufficiently large k, ϕ^{kd} will be a morphism that, depending on the letter σ to perform recursive applications of the morphism on, will always generate strings over some alphabet Σ_σ and $\phi^{kd}(\sigma')$ contains all symbols of Σ_σ for all $\sigma' \in \Sigma_\sigma$. As proving the claim for ϕ^{kd} implies it for ϕ as well, the assumption mentioned above does not harm the generality.

We now proceed to prove that for any morphism ϕ of rank r as described above, any positive integer $n \in \mathbb{N}$, and any positive constant $0 < \delta < 1$, there exists $m \in \mathbb{N}$ so that

$$LCS(\phi^m(a), \phi^m(b)) \geq \left[1 - \left(1 - \frac{1}{|\Sigma|^2 r}\right)^n - \delta\right] \cdot r^m$$

for any $a, b \in \Sigma$ where ϕ^m represents m consecutive applications of morphism ϕ and $LCS(.,.)$ denotes the longest common substring.

Having such claim proved, one can take $\delta = (1-\varepsilon)/2$ and n large enough so that m applications of ϕ over any pair of symbols entail strings with a longest common substring that is of a fraction larger than $1-(1-\varepsilon) = \varepsilon$ in terms of the length of those strings. Then, for any string $S \in \Sigma^*$, one can take two arbitrary consecutive symbols of $\phi(S)$ like $S[i]$ and $S[i+1]$. Applying morphism ϕ, m more times on $\phi(S)$ makes the corresponding intervals of $\phi^{m+1}(S)$ have an edit distance that is smaller than $1-\varepsilon$ fraction of their combined lengths. This shows that $\phi^{m+1}(S)$ is not an ε-synchronization string and finishes the proof.

Finally, we prove the claim by induction on n. For the base case of $n = 1$, given the assumption of all members of Σ appearing in $\phi(\sigma)$ for all $\sigma \in \Sigma$, $\phi(a)$ and $\phi(b)$ have a non-empty common subsequence. This gives that

$$LCS(\phi(a), \phi(b)) \geq 1 = \left[1 - \left(1 - \frac{1}{r}\right)\right] \cdot r > \left[1 - \left(1 - \frac{1}{|\Sigma|^2 r}\right) - \delta\right] \cdot r.$$

Therefore, choosing $m = 1$ finishes the induction base.

We now prove the induction step. Note that by induction hypothesis, for some given n, one can find m_1 such that

$$LCS(\phi^{m_1}(a), \phi^{m_1}(b)) \geq \left[1 - \left(1 - \frac{1}{|\Sigma|^2 r}\right)^n - \frac{\delta}{2}\right] \cdot r^{m_1}.$$

Now, let $m_2 = \left\lceil \log_r \frac{2}{\delta} \right\rceil$. Consider $\phi^{m_2}(a)$ and $\phi^{m_2}(b)$. Note that among all possible pairs of symbols from Σ^2, one appears at least $\frac{r^{m_2}}{|\Sigma|^2}$ times in respective positions of $\phi^{m_2}(a)$ and $\phi^{m_2}(b)$. Let (a', b') be such pair. As $\phi(a')$ and $\phi(b')$ contain all symbols of Σ, one can

take one specific occurrence of a fixed arbitrary symbol $\sigma \in \Sigma$ in all appearances of the pair $\phi(a')$ and $\phi(b')$ to find a common subsequence of size $\frac{r^{m_2}}{|\Sigma|^2} = \frac{r^{m_2+1}}{|\Sigma|^2 r}$ or more between $\phi^{m_2+1}(a)$ and $\phi^{m_2+1}(b)$ (See Figure 10.6).

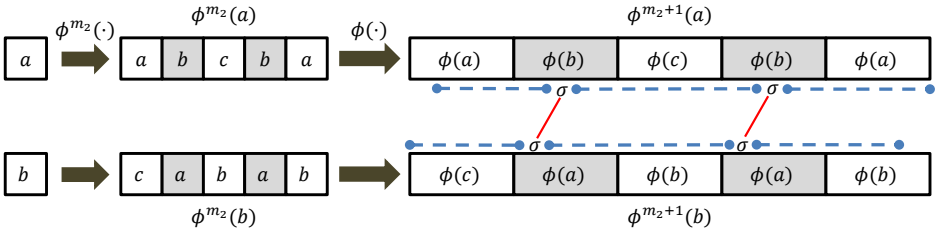


Figure 10.6: Induction step in Theorem 10.6.1; Most common pair $(a', b') = (b, a)$.

Note that one can apply the morphism ϕ further times over $\phi^{m_2+1}(a)$ and $\phi^{m_2+1}(b)$ and such common subsequence will still be preserved; However, one might be able to increase the size of it by adding new elements to the common subsequence from equal length pairs of intervals between current common subsequence elements (denoted by blue dashed line in Figure 10.6). The total length of such intervals is

$$1 - \frac{1}{|\Sigma|^2 r} - \frac{r}{r^{m_2+1}} = 1 - \frac{1}{|\Sigma|^2 r} - \frac{\delta}{2}$$

or more. In fact, using the induction hypothesis, by applying the morphism m_1 more times, one can get the following for $m = m_1 + m_2 + 1$.

$$\begin{aligned} LCS(\phi^m(a), \phi^m(b)) &\geq \left[\frac{1}{|\Sigma|^2 r} + \left(1 - \left(1 - \frac{1}{|\Sigma|^2 r}\right)^n - \frac{\delta}{2}\right)\right. \\ &\quad \left. \cdot \left(1 - \frac{1}{|\Sigma|^2 r} - \frac{\delta}{2}\right)\right] r^m \\ &\geq \left[1 - \left(1 - \frac{1}{|\Sigma|^2 r}\right)^{n+1} - \delta\right] r^m \end{aligned}$$

This completes the induction step and finishes the proof. □

10.6.2 Synchronization Strings over Alphabets of Size Four

In this section, we show that synchronization strings of arbitrary length exist over alphabets of size four. In order to do so, we first introduce the notion of *weak ε-synchronization strings*. This weaker notion is very similar to the synchronization string property except the edit distance requirement is rounded down.

Definition 10.6.2 (weak ε-synchronization strings)**.** *String S of length n is a weak ε-synchronization string if for every $1 \leq i < j < k \leq n$,*

$$\mathrm{ED}(S[i, j), S[j, k)) \geq \lfloor (1 - \varepsilon)(k - i) \rfloor.$$

We start by showing that binary weak ε-synchronization strings exist for some $\varepsilon < 1$.

Binary Weak ε-Synchronization Strings

Here we prove that an infinite binary weak ε-synchronization string exists. The main idea is to take a synchronization string over some large alphabet and convert it to a binary weak synchronization string by mapping each symbol of that large alphabet to a binary string and separating each binary encoded block with a block of the form 0^k1^k.

Theorem 10.6.3. *There exists a constant $\varepsilon < 1$ and an infinite binary weak ε-synchronization string.*

Proof. Take some arbitrary $\varepsilon' \in (0, 1)$. According to the discussions from Chapter 3, there exists an infinite ε'-synchronization string S over a sufficiently large alphabet Σ. Let $k = \lceil \log |\Sigma| \rceil$. Translate each symbol of S into k binary bits, and separate the translated k-blocks with 0^k1^k. We claim that this new string T is a weak ε-synchronization binary string for some $\varepsilon < 1$.

First, call a translated k-length symbol followed by 0^k1^k a *full block*. Call any other (possibly empty) substring a *half block*. Then any substring of T is a half-block followed by multiple full blocks and ends with a half block.

Let A and B be two consecutive substrings in T. Without loss of generality, assume $|A| \leq |B|$ (because edit distance is symmetric). Let M be a longest common subsequence between A and B. Partition blocks of B into the following 4 types of blocks:

1. Full blocks that match completely to another full block in A.
2. Full blocks that match completely but not to just 1 full block in A.
3. Full blocks where not all bits within are matched.
4. Half blocks.

The key claim is that the $3k$ elements in B which are matched to a type-2 block in A are not contiguous and, therefore, there is at least one unmatched symbol in B surrounded by them. To see this, assume by contradiction that all letters of some type-2 block in A are matched contiguously. The following simple analysis over 3 cases contradicts this assumption:

- Match starts at middle of some translated k-length symbol, say position $p \in [2, k]$. Then the first 1 of 1^k in A will be matched to the $(k - p + 2)$-th 0 of 0^k in B, contradiction.
- Match starts at 0-portion of 0^k1^k block, say at the p-th 0. Then the p-th 1 of 1^k in A will be matched to the first 0 of 0^k in B, contradiction.
- Match starts at 1-portion of 0^k1^k block, say at the p-th 1. Then the p-th 0 of 0^k in A will be matched to the first 1 of 1^k in B, contradiction.

Let the number of type-i blocks in B be t_i. For every type-2 block, there is an unmatched letter in A between its first and last matches. Hence, $|A| \geq |M| + t_2$. For every type-3 block, there is an unmatched letter in B within. Hence, $|B| \geq |M| + t_3$. Therefore, $|A| + |B| \geq 2|M| + t_2 + t_3$.

Since $|A| \leq |B|$, the total number of full blocks in both A and B is at most $2(t_1 + t_2 + t_3) + 1$. (the additional $+1$ comes from the possibility that the two half-blocks in B allows for one extra full block in A) Note t_1 is a matching between the full blocks in A and the full blocks in B. So due to the ε'-synchronization property of S, we obtain the following.

$$t_1 \leq \frac{\varepsilon'}{2}\left(2(t_1 + t_2 + t_3) + 1\right) \implies t_1 \leq \frac{\varepsilon'}{1-\varepsilon'}(t_2 + t_3) + \frac{\varepsilon'}{2(1-\varepsilon')}$$

Furthermore, $t_1 + t_2 + t_3 + 2 > \frac{|B|}{3k} \geq \frac{|A|+|B|}{6k}$. This, along with the above inequality, implies the following.

$$\frac{1}{1-\varepsilon'}(t_2 + t_3) + \frac{4 - 3\varepsilon'}{2(1-\varepsilon')} > \frac{|A| + |B|}{6k}.$$

The edit distance between A and B is

$$\begin{aligned} ED(A, B) &= |A| + |B| - 2|M| \geq t_2 + t_3 \\ &> \frac{1-\varepsilon'}{6k}(|A| + |B|) - \frac{4 - 3\varepsilon'}{2} > \frac{1-\varepsilon'}{6k}(|A| + |B|) - 2. \end{aligned}$$

Set $\varepsilon = 1 - \frac{1-\varepsilon'}{18k}$. If $|A| + |B| \geq \frac{1}{1-\varepsilon}$, then

$$\begin{aligned} \frac{1-\varepsilon'}{6k}(|A| + |B|) - 2 &\geq \left(\frac{1-\varepsilon'}{6k} - 2(1-\varepsilon)\right)(|A| + |B|) \\ &= (1-\varepsilon)(|A| + |B|) \geq \lfloor (1-\varepsilon)(|A| + |B|) \rfloor. \end{aligned}$$

As weak ε-synchronization property trivially holds for $|A| + |B| < \frac{1}{1-\varepsilon}$, this will prove that T is a weak ε-synchronization string. □

ε-Synchronization Strings over Alphabets of Size Four

A corollary of Theorem 10.6.3 is the existence of infinite synchronization strings over alphabets of size four. Here we make use of the fact that infinite ternary square-free strings exist, which was proven in previous work [Thu06]. We then modify such a string to fulfill the synchronization string property, using the existence of an infinite binary weak synchronization string.

Theorem 10.6.4. *There exists some $\varepsilon \in (0, 1)$ and an infinite ε-synchronization string over an alphabet of size four.*

Proof. Take an infinite ternary square-free string T over alphabet $\{1, 2, 3\}$ [Thu06] and some $\varepsilon \in \left(\frac{11}{12}, 1\right)$. Let S be an infinite weak binary $\varepsilon' = (12\varepsilon - 11)$-synchronization string. Consider the string W that is similar to T except that the i-th occurrence of symbol 1 in

T is replaced with symbol 4 if $S[i] = 1$. Note W is still square-free. We claim W is an ε-synchronization string as well.

Let $A = W[i, j), B = W[j, k)$ be two consecutive substrings of W. If $k - i < 1/(1 - \varepsilon)$, then $ED(A, B) \geq 1 > (1 - \varepsilon)(k - i)$ by square-freeness.

Otherwise, $k - i \geq 1/(1 - \varepsilon) \geq 12$. Consider all occurrences of 1 and 4 in A and B, which form consecutive subsequences A_s and B_s of S respectively. Note that $|A_s| + |B_s| \geq (k - i - 3)/4$, because, by square-freeness, there cannot be a length-4 substring consisting only of 2's and 3's in W.

By weak synchronization property,

$$\begin{aligned} ED(A_s, B_s) &\geq \lfloor (1 - \varepsilon')(|A_s| + |B_s|) \rfloor \\ &\geq \lfloor 3(1 - \varepsilon)(k - i - 3) \rfloor > 3(1 - \varepsilon)(k - i) - 9(1 - \varepsilon) - 1 \\ &\geq (1 - \varepsilon)(k - i), \end{aligned}$$

and hence, $ED(A, B) \geq ED(A_s, B_s) \geq (1-\varepsilon)(k-i)$. Therefore, W is an ε-synchronization string. □

10.7 Lower-bounds for ε in Infinite ε-Synchronization Strings

It is known from Section 10.6.2 that infinite synchronization strings exist over alphabet sizes $|\Sigma| \geq 4$. A natural question to ask is the optimal value of ε for each such $|\Sigma|$. Formally, we seek to discover

$$B_k = \inf\{\varepsilon : \text{there exists an infinite } \varepsilon\text{-synchronization string with } |\Sigma| = k\}$$

for small values of k. To that end, a program was written to find an upper bound for B_k for $k \leq 6$. The program first fixes an ε, then exhaustively enumerates all possible ε-synchronization strings over an alphabet size of k by increasing length. If the program terminates, then this ε is a proven lower bound for B_k. Among every pair of consecutive substrings in each checked string that failed the ε-synchronization property, we find the one that has the lowest edit distance relative to their total length and such fraction would be a lower-bound for B_k as well. Such experimentally obtained lower-bounds for alphabets of size 3, 4, 5, and 6 are listed in Table 10.1.

k	$B_k \geq \cdot$
3	12/13
4	10/13
5	2/3
6	18/29

Table 10.1: Computationally proven lower-bounds of B_k

Chapter 11

Concluding Remarks

In this dissertation, we addressed several fundamental questions regarding coding for synchronization errors – questions which were extensively studied or even completely answered for coding against ordinary symbol substitution errors but were left open for several decades in the context of synchronization errors. While this thesis sheds light on several aspects of synchronization coding, much has remained unknown. The open questions span from fundamental theoretical properties of synchronization channels to more practical challenges rising in real computer systems afflicted by such errors. In this chapter, we review the contributions of the thesis and remark some remaining questions that can inspire future work on synchronization coding.

Most of the contributions of this thesis originated from the introduction of an indexing based coding scheme in Chapter 3. We introduced synchronization strings as simple yet very powerful mathematical objects that are very helpful when dealing with synchronization errors in communications. In particular, we used them to efficiently transform insertion and deletion errors into much simpler Hamming-type errors in several communication problems.

An important question that this dissertation attempted to address was the characterization of the trade-off between the rate and distance in synchronization coding schemes in various settings. In Chapter 3, we provided families of near-MDS insertion-deletion correcting codes, i.e., for any $\delta \in (0, 1)$ and $\varepsilon > 0$, we gave a family of codes with relative distance δ and rate $1 - \delta - \varepsilon$. Furthermore, these codes are efficient and were shown to have an asymptotically optimal dependence of alphabet size on parameter ε. Later, in Chapter 9, we improved the code construction to achieve near-linear time decoding. This was achieved via indexing schemes that facilitated approximating the edit distance of the indexed string to any other string in near-linear time and within a $1 + \varepsilon$ factor. Finding near-MDS insertion-deletion codes that are decodable in linear time remains an interesting open question. Such code would be the equivalent of the breakthrough of Spielman [Spi96] for ECCs.

In Chapter 4, we extended this result to the list-decoding setting. We showed that a similar indexing scheme can yield families of code that achieve near-optimal trade-off between the rate and distance given that the alphabet size can be an arbitrarily large constant. More precisely, for any $\delta \in (0, 1)$, $\gamma > 0$, and $\varepsilon > 0$, we gave a family of efficient

codes that are list-decodable from δ-fraction of deletions and γ fraction of insertions and achieve a rate of at least $1 - \delta - \varepsilon$.

In Chapter 5, we started to factor in the alphabet size in our study of the rate-distance trade-off for the list-decoding setting. We were able to fully identify the set of all pairs (γ, δ) where positive-rate list-decodable codes correcting δ-fraction of deletions and γ fraction of insertions exist. We furthermore provided explicit and efficient codes that are optimally resilient. As the next natural step, we provided several upper- and lower-bounds on the worst-case list-decoding capacity in Chapter 6. The exact characterization of the worst-case list-decoding capacity of insertion-deletion channels remains open. Similar question for unique-decoding has remained open as well. As mentioned in Chapter 5, even the optimal error resilience for uniquely decodable insertion-deletion codes is unknown.

In Chapter 7, we showed that our synchronization string based indexing method can be extended to fully simulate an ordinary substitution channel over a given insertion-deletion channel (i.e., without any delay for repositioning). This was much stronger than constructing insdel codes and allows us to completely hide the existence of synchronization errors in many applications that go beyond insertion-deletion codes, such as, settings with feedback, real-time control, or interactive communications. This finding lead to new interactive coding schemes for the setting with synchronization errors including the first efficient scheme and the first scheme with good (and in fact likely near-optimal) communication rate for small error fractions.

Chapter 7 also showed that the code construction methods presented in Chapter 3 can be ported to the setting of binary codes to derive binary codes that tolerate a δ fraction of synchronization errors while achieving a rate of $1 - O(\sqrt{\delta \log(1/\delta)})$.

In Chapter 8, we provided several highly parallel, deterministic linear time constructions for ε-synchronization strings. These constructions provide highly-explicit infinite ε-synchronization strings in which the ith symbol can be computed deterministically in $O(\log i)$ time. Chapter 8 also gives strengthened versions of the ε-synchronization string property which comes with very fast local repositioning procedures that improve the time and space complexities of channel simulations and interactive coding schemes from Chapter 7 as well as codes correcting from block transposition and replication errors.

In Chapter 9, we employed a similar indexing scheme with a specific pseudo-random string that enhances the global repositioning algorithm presented in Section 3.5.3 by decreasing its time complexity to near-linear time. This reduced the decoding complexity of codes from Chapters 3 and 4.

Finally, in Chapter 10, we studied the combinatorial properties of synchronization strings and addressed several extremal questions regarding them. This chapter leaves a few open questions regarding synchronization strings as combinatorial objects of their own interest, such as the minimal alphabet size over which synchronization strings exist.

Besides open questions set forth above, there are plenty of questions that have been studied over decades for coding against Hamming-type errors that one can address in the context of synchronization coding. The study of *linear* synchronization code is one such example. A very recent work [CGHL20] have studied linear and affine synchronization codes and have presented asymptotically good linear insertion-deletion codes using synchronization strings. Another such question would be to investigate systematic codes

for synchronization errors. While synchronization-based constructions presented in this work do not yield systematic synchronization codes, it is an interesting question to investigate systematic synchronization codes. It may be possible, as shown to some extent in [CGHL20], to develop resistance against synchronization errors with careful placement of non-systematic symbols in a codeword. One can also think about formalizing local decodability qualities for synchronization codes. Furthermore, some of the tools and techniques presented in this document may be applicable to other relevant problems. A number of recent works have already demonstrated the applicability of our techniques to problems like document exchange [CJLW18, Hae19], trace reconstruction [BLS19], or coding against block errors [CJLW19]. We hope that the ideas and techniques put forward in this dissertation inspire further research on synchronization coding and the progress made by this body of work along with the recent renewed interest of the community in insertion-deletion codes lead to new directions in the fundamental study of timing issues in digital communication.

Bibliography

[ABW15] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 59–78, 2015. 9.1.3

[AEL95] Noga Alon, Jeff Edmonds, and Michael Luby. Linear time erasure codes with nearly optimal recovery. In *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 512–519, 1995. 8.1.3, 9.1.1

[AGPFC11] Khaled AS Abdel-Ghaffar, Filip Paluncic, Hendrik C Ferreira, and Willem A Clarke. On Helberg's generalization of the Levenshtein code for multiple deletion/insertion error correction. *IEEE Transactions on Information Theory*, 58(3):1804–1808, 2011. 1.2

[AGS16] Shweta Agrawal, Ran Gelles, and Amit Sahai. Adaptive protocols for interactive communication. In *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, pages 595–599, 2016. 7.1

[AHWW16] Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 375–388, 2016. 9.1.3

[AK12] Alexandr Andoni and Robert Krauthgamer. The smoothed complexity of edit distance. *ACM Transactions on Algorithms (TALG)*, 8(4):44:1–44:25, 2012. 9.1.3

[AKO10] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 377–386, 2010. 9.1.3

[AO12] Alexandr Andoni and Krzysztof Onak. Approximating edit distance in near-linear time. *SIAM Journal on Computing*, 41(6):1635–1648, 2012. 9.1.3

[APP13] Maria Axenovich, Yury Person, and Svetlana Puzynina. A regularity lemma and twins in words. *Journal of Combinatorial Theory, Series A*, 120(4):733–743, 2013. 10.5

[Ari09] Erdal Arikan. Channel polarization: a method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Transactions on Information Theory*, 55(7):3051–3073, 2009. 5.1.4

[AWW14] Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP), Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 39–51. Springer, 2014. 9.1.3

[BE17] Mark Braverman and Klim Efremenko. List and unique coding for interactive communication in the presence of adversarial noise. *SIAM Journal on Computing*, 46(1):388–428, 2017. 7.1

[Bec84] J Beck. An application of Lovász local lemma: there exists an infinite 01-sequence containing no near identical intervals. In *Finite and Infinite Sets*, pages 103–107. Elsevier, 1984. 8.3.1

[BEG+18] Mahdi Boroujeni, Soheil Ehsani, Mohammad Ghodsi, MohammadTaghi HajiAghayi, and Saeed Seddighin. Approximating edit distance in truly subquadratic time: Quantum and MapReduce. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1170–1189, 2018. 9.1.3, 9.4.1

[BES06] Tuğkan Batu, Funda Ergun, and Cenk Sahinalp. Oblivious string embeddings and edit distance approximations. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 792–801, 2006. 9.1.3

[BG16] Boris Bukh and Venkatesan Guruswami. An improved bound on the fraction of correctable deletions. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1893–1901, 2016. 5.1.1

[BGH+16] Meinolf Blawat, Klaus Gaedke, Ingo Huetter, Xiao-Ming Chen, Brian Turczyk, Samuel Inverso, Benjamin W Pruitt, and George M Church. Forward error correction for DNA data storage. *Procedia Computer Science*, 80:1011–1022, 2016. 1

[BGH17] Boris Bukh, Venkatesan Guruswami, and Johan Håstad. An improved bound on the fraction of correctable deletions. *IEEE Transactions on Information Theory*, 63(1):93–103, 2017. 4.1.1, 5.1.1, 5.1.1, 5.1.3, 5.1.3

[BGMO17] Mark Braverman, Ran Gelles, Jieming Mao, and Rafail Ostrovsky. Coding for interactive communication correcting insertions and deletions. *IEEE Transactions on Information Theory*, 63(10):6256–6270, 2017. 1.4, 3.1, 3.1.2, 3.1.3,

3.2, 3.4, 3.4.2, 3.5.7, 7, 7.1, 7.1.1, 7.1.1, 7.1.2, 7.2, 7.2, 7.6, 7.6.2, 7.6.3, 7.6.4, 7.6.1, 7.6.5, 7.6.1, 7.6.6, 7.6.1, 7.6.2, 7.6.12, 8.1.5

[BGN+18] Jaroslaw Blasiok, Venkatesan Guruswami, Preetum Nakkiran, Atri Rudra, and Madhu Sudan. General strong polarization. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 485–492, 2018. 5.1.4

[BGZ18] Joshua Brakensiek, Venkatesan Guruswami, and Samuel Zbarsky. Efficient low-redundancy codes for correcting multiple deletions. *IEEE Transactions on Information Theory*, 64(5):3403–3410, 2018. 1.2

[BHN08] Paul Beame and Dang-Trinh Huynh-Ngoc. On the value of multiple read/write streams for approximating frequency moments. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 499–508, 2008. 10.5

[BI18] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM Journal on Computing*, 47(3):1087–1097, 2018. 3.1, 8.1.3, 9.1.3

[BK15] Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 79–97, 2015. 9.1.3

[BK18] Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1216–1235, 2018. 9.1.3

[BKN14] Zvika Brakerski, Yael Tauman Kalai, and Moni Naor. Fast interactive coding against adversarial noise. *Journal of the ACM (JACM)*, 61(6):35, 2014. 7.1

[BLC+16] James Bornholt, Randolph Lopez, Douglas M Carmean, Luis Ceze, Georg Seelig, and Karin Strauss. A DNA-based archival storage system. *ACM SIGARCH Computer Architecture News*, 44(2):637–649, 2016. 1

[BLS19] Joshua Brakensiek, Ray Li, and Bruce Spang. Coded trace reconstruction in a constant number of traces. *arXiv preprint arXiv:1908.03996*, 2019. 11

[BM14] Boris Bukh and Jie Ma. Longest common subsequences in sets of words. *SIAM Journal on Discrete Mathematics*, 28(4):2042–2049, 2014. 5, 5.1.3, 5.1.4

[BN13] Zvika Brakerski and Moni Naor. Fast algorithms for interactive coding. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 443–456, 2013. 1.4, 7.1, 8, 8.1.5

[BNT+19] Gilles Brassard, Ashwin Nayak, Alain Tapp, Dave Touchette, and Falk Unger. Noisy interactive quantum communication. *SIAM Journal on Computing*, 48(4):1147–1195, 2019. 7.1

[BR14] Mark Braverman and Anup Rao. Toward coding for maximum errors in interactive communication. *IEEE Transactions on Information Theory*, 60(11):7248–7255, 2014. 7.1, 8.1.5

[Bra12] Mark Braverman. Towards deterministic tree code constructions. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (ITCS)*, pages 161–167, 2012. 7.1.1, 7.6.2

[BTK12] Zvika Brakerski and Yael Tauman Kalai. Efficient interactive coding against adversarial noise. In *Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 160–166, 2012. 7.1, 8.1.5

[BYJKK04] Ziv Bar-Yossef, TS Jayram, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 550–559, 2004. 9.1.3

[BZ16] Boris Bukh and Lidong Zhou. Twins in words and long common subsequences in permutations. *Israel Journal of Mathematics*, 213(1):183–209, 2016. 10.5, 10.5.1

[CDG+18] Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucky, and Michael Saks. Approximating edit distance within constant factor in truly sub-quadratic time. In *Proceedings of the 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 979–990, 2018. 9.1.3

[CGH13] Karthekeyan Chandrasekaran, Navin Goyal, and Bernhard Haeupler. Deterministic algorithms for the Lovász local lemma. *SIAM Journal on Computing*, 42(6):2132–2155, 2013. 3.5.2, 8.3.2, 8.3.2, 8.3.2, 10.4.1, 10.4.1

[CGHL20] Kuan Cheng, Venkatesan Guruswami, Bernhard Haeupler, and Xin Li. Efficient linear and affine codes for correcting insertions/deletions. *arXiv preprint arXiv:2007.09075*, 2020. 11

[CGK12] George M Church, Yuan Gao, and Sriram Kosuri. Next-generation digital information storage in DNA. *Science*, 337(6102):1628–1628, 2012. 1

[CHL+19] Kuan Cheng, Bernhard Haeupler, Xin Li, Amirbehshad Shahrasbi, and Ke Wu. Synchronization strings: highly efficient deterministic constructions over small alphabets. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2185–2204, 2019. 1.5

[CJLW18] Kuan Cheng, Zhengzhong Jin, Xin Li, and Ke Wu. Deterministic document exchange protocols, and almost optimal binary codes for edit errors. In *Proceedings of the 59th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 200–211, 2018. 7.1.1, 11

[CJLW19] Kuan Cheng, Zhengzhong Jin, Xin Li, and Ke Wu. Block edit errors with transpositions: Deterministic document exchange protocols and almost optimal binary codes. In *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 132 of *LIPIcs*, pages 37:1–37:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. 8.1.3, 11

[CKK72] Václav Chvátal, David A. Klarner, and Donald E. Knuth. Selected combinatorial research problems. Technical report, Computer Science Department, Stanford University, 1972. 9.1.3

[CLW17] Kuan Cheng, Xin Li, and Ke Wu. Synchronization strings: Efficient and fast deterministic constructions over small alphabets. *arXiv preprint arXiv:1710.07356*, 2017. 8.1.1

[CR+16] Serina Camungol, Narad Rampersad, et al. Avoiding approximate repetitions with respect to the longest common subsequence distance. *Involve, a Journal of Mathematics*, 9(4):657–666, 2016. 10.1.1, 10.1.2

[CR20] Mahdi Cheraghchi and João Ribeiro. An overview of capacity results for synchronization channels. *IEEE Transactions on Information Theory*, 2020. 1.2

[Cro82] Max Crochemore. Sharp characterizations of squarefree morphisms. *Theoretical Computer Science*, 18(2):221–226, 1982. 10.1.1, 10.6.1

[Dan94] Vladimír Dančík. *Expected length of longest common subsequences*. PhD thesis, University of Warwick, 1994. 5.1.1

[DP95] Vlado Dančík and Mike Paterson. Upper bounds for the expected length of a longest common subsequence of two binary sequences. *Random Structures & Algorithms*, 6(4):449–458, 1995. 5.1.1

[Eli57] Peter Elias. List decoding for noisy channels. *Technical Report 335, Research Laboratory of Electronics, MIT*, 1957. 4.1.1

[ERB16] Klim Efremenko, Gelles Ran, and Haeupler Bernhard. Maximal noise in interactive communication over erasure channels and channels with feedback. *IEEE Transactions on Information Theory*, 62(8):4575–4588, 2016. 7.1

[FGOS15] Matthew Franklin, Ran Gelles, Rafail Ostrovsky, and Leonard J. Schulman. Optimal coding for streaming authentication and interactive communication. *IEEE Transactions on Information Theory*, 61(1):133–145, 2015. 7.1, 8.1.5

[Gaw12] Pawel Gawrychowski. Faster algorithm for computing the edit distance between slp-compressed strings. In *Proceedings of the 19th International Symposium on String Processing and Information Retrieval (SPIRE)*, volume 7608 of *Lecture Notes in Computer Science*, pages 229–236. Springer, 2012. 9.1.3

[GBC+13] Nick Goldman, Paul Bertone, Siyuan Chen, Christophe Dessimoz, Emily M LeProust, Botond Sipos, and Ewan Birney. Towards practical, high-capacity, low-maintenance information storage in synthesized DNA. *Nature*, 494(7435):77, 2013. 1

[GDR+63] SW Golomb, J Davey, I Reed, H Van Trees, and J Stiffler. Synchronization. *IEEE Transactions on Communications Systems*, 11(4):481–491, 1963. 1

[Gel17] Ran Gelles. Coding for interactive communication: A survey. *Foundations and Trends in Theoretical Computer Science*, 13(1–2):1–157, 2017. 3.1

[GH14] Mohsen Ghaffari and Bernhard Haeupler. Optimal error rates for interactive coding II: Efficiency and list decoding. In *Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 394–403, 2014. 3.1, 7.1, 7.1.1, 7.5, 7.5.1, 7.5, 8.1.5, 8.4, 8.7, 8.7.1

[GH17a] Ran Gelles and Bernhard Haeupler. Capacity of interactive communication over erasure channels and channels with feedback. *SIAM Journal on Computing*, 46(4):1449–1472, 2017. 3.1, 7.1

[GH17b] Shafi Goldwasser and Dhiraj Holden. The complexity of problems in P given correlated instances. In *Proceedings of the 8th Innovations in Theoretical Computer Science Conference (ITCS)*, volume 67 of *LIPIcs*, pages 13:1–13:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. 9.1.3

[GHS14] Mohsen Ghaffari, Bernhard Haeupler, and Madhu Sudan. Optimal error rates for interactive coding I: Adaptivity and other settings. In *Proceedings of the 46th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 794–803, 2014. 3.1, 7.1

[GHS20] Venkatesan Guruswami, Bernhard Haeupler, and Amirbehshad Shahrasbi. Optimally resilient codes for list-decoding from insertions and deletions. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 524–537, 2020. 1.5

[GI01] Venkatesan Guruswami and Piotr Indyk. Expander-based constructions of efficiently decodable codes. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 658–667, 2001. 4.2.2, 8.1.3

[GI02] Venkatesan Guruswami and Piotr Indyk. Near-optimal linear-time codes for unique decoding and new list-decodable codes over smaller alphabets. In

Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC), pages 812–821, 2002. 4.2.2

[GI03] Venkatesan Guruswami and Piotr Indyk. Linear time encodable and list decodable codes. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC)*, pages 126–135, 2003. 4.2.2

[GI04] Venkatesan Guruswami and Piotr Indyk. Linear-time list decoding in error-free settings. In *Proceedings of the 31st International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 695–707. Springer, 2004. 4.2.2

[GI05] Venkatesan Guruswami and Piotr Indyk. Linear-time encodable/decodable codes with near-optimal rate. *IEEE Transactions on Information Theory*, 51(10):3393–3400, 2005. 3.1, 3.1.5, 3.3.1, 3.3.3, 3.3.1, 8.1.3, 8.3.4, 8.3.11, 8.5.1, 9.1.1, 9.7.2, 9.7.2, 10.4.2, 10.4.11

[GL16] Venkatesan Guruswami and Ray Li. Efficiently decodable insertion/deletion codes for high-noise and high-rate regimes. In *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, pages 620–624, 2016. 3.1, 3.1.3, 3.1.5, 4.1.1, 5.1.1, 5.1.1, 5.1.3, 8.4, 9.1.2

[GL20] Venkatesan Guruswami and Ray Li. Coding against deletions in oblivious and online models. *IEEE Transactions on Information Theory*, 66(4):2352–2374, 2020. 5.1.3

[GMS11] Ran Gelles, Ankur Moitra, and Amit Sahai. Efficient and explicit coding for interactive communication. In *Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 768–777, 2011. 7.1

[GMS14] Ran Gelles, Ankur Moitra, and Amit Sahai. Efficient coding for interactive communication. *IEEE Transactions on Information Theory*, 60(3):1899–1913, 2014. 7.1, 8.1.5

[GR06] Venkatesan Guruswami and Atri Rudra. Explicit capacity-achieving list-decodable codes. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 1–10, 2006. 8.4

[GR08] Venkatesan Guruswami and Atri Rudra. Explicit codes achieving list decoding capacity: Error-correction with optimal redundancy. *IEEE Transactions on Information Theory*, 54(1):135–150, 2008. 3.1.5, 4.1.2, 4.2.2

[GS99] Venkatesan Guruswami and Madhu Sudan. Improved decoding of reed-solomon and algebraic-geometry codes. *IEEE Transactions on Information Theory*, 45(6):1757–1767, 1999. 4.2.2

[GS18] Ryan Gabrys and Frederic Sala. Codes correcting two deletions. *IEEE Transactions on Information Theory*, 65(2):965–974, 2018. 1.2

[GV15] Venkatesan Guruswami and Ameya Velingker. An entropy sumset inequality and polynomially fast convergence to shannon capacity over all alphabets. In *Proceedings of the 30th Conference on Computational Complexity (CCC)*, volume 33 of *LIPIcs*, pages 42–57. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. 3.1.5

[GW11] Venkatesan Guruswami and Carol Wang. Optimal rate list decoding via derivative codes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (RANDOM/APPROX)*, pages 593–604. Springer, 2011. 4.2.2

[GW17] Venkatesan Guruswami and Carol Wang. Deletion codes in the high-noise and high-rate regimes. *IEEE Transactions on Information Theory*, 63(4):1961–1970, 2017. 1.4, 3.1, 3.1.5, 4.1.1, 5.1.1, 5.1.1, 5.1.3, 5.1.4, 7, 7.1.1, 8.4, 9.1.2

[GX13] Venkatesan Guruswami and Chaoping Xing. List decoding Reed-Solomon, Algebraic-Geometric, and Gabidulin subcodes up to the Singleton bound. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC)*, pages 843–852, 2013. 4.2.2, 5.1.2

[GX15] Venkatesan Guruswami and Patrick Xia. Polar codes: Speed of polarization and polynomial gap to capacity. *IEEE Transactions on Information Theory*, 61(1):3–16, 2015. 3.1.5

[GX17] Venkatesan Guruswami and Chaoping Xing. Optimal rate list decoding over bounded alphabets using algebraic-geometric codes. *arXiv preprint arXiv:1708.01070*, 2017. 4.1.2, 4.2.2, 4.3.4

[Hae14] Bernhard Haeupler. Interactive channel capacity revisited. In *Proceedings of the 55th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 226–235, 2014. 3.1, 7.1, 7.1.1, 7.5, 7.5.2, 7.5.3, 7.5, 7.5, 7.5.5, 7.5, 7.5, 8.1.5

[Hae19] Bernhard Haeupler. Optimal document exchange and new codes for insertions and deletions. In *Proceedings of the 60th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 334–347, 2019. 7.1.1, 11

[HF02] Albertus SJ Helberg and Hendrik C Ferreira. On multiple insertion/deletion correcting codes. *IEEE Transactions on Information Theory*, 48(1):305–308, 2002. 1.2

[Hir77] Daniel S. Hirschberg. Algorithms for the longest common subsequence problem. *Journal of the ACM (JACM)*, 24(4):664–675, 1977. 9.1.3

[HRS19] Bernhard Haeupler, Aviad Rubinstein, and Amirbehshad Shahrasbi. Near-linear time insertion-deletion codes and (1+ε)-approximating edit distance via indexing. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 697–708, 2019. 1.5

[HRZW19] Brett Hemenway, Noga Ron-Zewi, and Mary Wootters. Local list recovery of high-rate tensor codes and applications. *SIAM Journal on Computing*, Special Section FOCS 2017:157–195, 2019. 4.2.2, 4.2.4, 4.3, 4.3.4, 8.1.3, 9.2.2, 9.6, 9.6.2

[HS77] James W Hunt and Thomas G Szymanski. A fast algorithm for computing longest common subsequences. *Communications of the ACM*, 20(5):350–353, 1977. 9.1.3, 9.3.2, 9.3.2, 9.3.6, 9.4, 9.4.1, 9.5.1

[HS17] Bernhard Haeupler and Amirbehshad Shahrasbi. Synchronization strings: Codes for insertions and deletions approaching the Singleton bound. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 33–46, 2017. 1.5

[HS18] Bernhard Haeupler and Amirbehshad Shahrasbi. Synchronization strings: Explicit constructions, local decoding, and applications. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 841–854, 2018. 1.5

[HS20] Bernhard Haeupler and Amirbehshad Shahrasbi. On the rate of list-decodable insertion-deletion codes, 2020. 1.5

[HSS11] Bernhard Haeupler, Barna Saha, and Aravind Srinivasan. New constructive aspects of the Lovász local lemma. *Journal of the ACM (JACM)*, 58(6):28, 2011. 3.4.1, 10.3

[HSS18] Bernhard Haeupler, Amirbehshad Shahrasbi, and Madhu Sudan. Synchronization strings: List decoding for insertions and deletions. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 107 of *LIPIcs*, pages 76:1–76:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. 1.5

[HSV18] Bernhard Haeupler, Amirbehshad Shahrasbi, and Ellen Vitercik. Synchronization strings: Channel simulations and interactive coding for insertions and deletions. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 107 of *LIPIcs*, pages 75:1–75:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. 1.5

[HW18] Brett Hemenway and Mary Wootters. Linear-time list recovery of high-rate expander codes. *Information and Computation*, 261:202–218, 2018. 4.2.2

[HY18] Tomohiro Hayashi and Kenji Yasunaga. On the list decodability of insertions and deletions. In *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, pages 86–90, 2018. 4.1.1, 5.1.1, 5.1.4

[KLM05] Marcos Kiwi, Martin Loebl, and Jiří Matoušek. Expected length of the longest common subsequence for large alphabets. *Advances in Mathematics*, 197(2):480–498, 2005. 5.1.1

[Kop15] Swastik Kopparty. List-decoding multiplicity codes. *Theory of Computing*, 11(5):149–182, 2015. 4.2.2

[KORS07] Dalia Krieger, Pascal Ochem, Narad Rampersad, and Jeffrey Shallit. Avoiding approximate squares. In *International Conference on Developments in Language Theory*, pages 278–289. Springer, 2007. 10.1.1, 10.1.2, 10.1.2, 10.6.1

[KR13] Gillat Kol and Ran Raz. Interactive channel capacity. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC)*, pages 715–724, 2013. 3.1, 7.1, 8.1.5

[KRR+19] Swastik Kopparty, Nicolas Resch, Noga Ron-Zewi, Shubhangi Saraf, and Shashwat Silas. On list recovery of high-rate tensor codes. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, volume 145 of *LIPIcs*, pages 68:1–68:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. 4.2.2

[Kus19] William Kuszmaul. Efficiently approximating edit distance between pseudorandom strings. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1165–1180, 2019. 9.1.3

[Lee57] John Leech. 2726. a problem on strings of beads. *The Mathematical Gazette*, 41(338):277–278, 1957. 10.1.1, 10.6.1

[Lev65] Vladimir Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii Nauk SSSR*, 163(4):845–848, 1965. English translation in *Soviet Physics Doklady*, 10(8):707–710, 1966. 1.2, 2.2, 3.1, 3.1.5, 5.1.1, 9.1.2

[Lev74] Vladimir I Levenshtein. Elements of coding theory. *Diskretnaya matematika i matematicheskie voprosy kibernetiki*, pages 207–305, 1974. 6.4.1

[LMS98] Gad M Landau, Eugene W Myers, and Jeanette P Schmidt. Incremental string comparison. *SIAM Journal on Computing*, 27(2):557–582, 1998. 9.1.3

[LTX19] Shu Liu, Ivan Tjuawinata, and Chaoping Xing. List decoding of insertion and deletion codes. *arXiv preprint arXiv:1906.09705*, 2019. 4.1.1

[Lub02] Michael Luby. LT codes. pages 271–282, 2002. 3.1.5

[Lue09] George S Lueker. Improved bounds on the average length of longest common subsequences. *JACM*, 56(3):17:1–17:38, 2009. 5.1.1

[LYC03] S-YR Li, Raymond W Yeung, and Ning Cai. Linear network coding. *IEEE Transactions on Information Theory*, 49(2):371–381, 2003. 3.1.5

[MBT10] Hugues Mercier, Vijay K Bhargava, and Vahid Tarokh. A survey of error-correcting codes for channels with symbol synchronization errors. *IEEE Communications Surveys & Tutorials*, 12(1):87–96, 2010. 1, 1.2, 3.1, 3.1.5

[Mit09] Michael Mitzenmacher. A survey of results for deletion channels and related synchronization channels. *Probability Surveys*, 6:1–33, 2009. 1, 1.2, 3, 3.1.5

[MP80] William J Masek and Michael S Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System sciences*, 20(1):18–31, 1980. 9.1.3

[MT10] Robin A. Moser and Gabor Tardos. A constructive proof of the general lovász local lemma. *Journal of the ACM (JACM)*, 57(2):11, 2010. 3.4.1, 10.3

[NN93] Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM Journal on Computing*, 22(4):838–856, 1993. 3.1.3, 3.5.2, 3.5.2

[OAC+17] Lee Organick, Siena Dumas Ang, Yuan-Jyue Chen, Randolph Lopez, Sergey Yekhanin, Konstantin Makarychev, Miklos Z. Racz, Govinda Kamath, Parikshit Gopalan, Bichlien Nguyen, Christopher Takahashi, Sharon Newman, Hsing-Yeh Parker, Cyrus Rashtchian, Kendall Stewart, Gagan Gupta, Robert Carlson, John Mulligan, Douglas Carmean, Georg Seelig, Luis Ceze, and Karin Strauss. Scaling up DNA data storage and random access retrieval. *BioRxiv*, 2017. 1

[RT06] Eran Rom and Amnon Ta-Shma. Improving the alphabet-size in expander-based code constructions. *IEEE Transactions on Information Theory*, 52(8):3695–3700, 2006. 3.3.1

[Rub18] Aviad Rubinstein. Approximating edit distance. `https://theorydish.blog/2018/07/20/approximating-edit-distance/`, 2018. 9.1.3

[Sch92] Leonard J. Schulman. Communication on noisy channels: A coding theorem for computation. In *Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 724–733, 1992. 7.1, 8.1.5

[Sch93] Leonard J. Schulman. Deterministic coding for interactive communication. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC)*, pages 747–756, 1993. 7.1, 7.6.1, 7.6.2

[Sch96] Leonard J. Schulman. Coding for interactive communication. *IEEE Transactions on Information Theory*, 42(6):1745–1756, 1996. 7.6.8, 7.6.2, 8.1.5

[Sch03] Leonard J. Schulman. Postscript to "Coding for interactive communication". [Online; accessed 17-March-2017], 2003. 7.1.1, 7.6.2

[She81] Robert Shelton. Aperiodic words on three symbols. *Journal für die Reine und Angewandte Mathematik*, 321:195–209, 1981. 10.1.1

[Slo02] Neil J. A Sloane. On single-deletion-correcting codes. *Codes and Designs*, 10:273–291, 2002. 1.2, 3.1.5

[Spi95] Daniel Alan Spielman. *Computationally efficient error-correcting codes and holographic proofs*. PhD thesis, Massachusetts Institute of Technology, 1995. 8.1.3

[Spi96] Daniel A Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Transactions on Information Theory*, 42(6):1723–1731, 1996. 3.1, 3.1.5, 9.1.1, 11

[SS82] Robert O Shelton and Raj P Soni. Aperiodic words on three symbols iii. *Journal für die Reine und Angewandte Mathematik*, 330:44–52, 1982. 10.1.1

[SS96] Michael Sipser and Daniel A Spielman. Expander codes. *IEEE Transactions on Information Theory*, 42(6):1710–1722, 1996. 8.1.3, 9.1.1

[Sud97] Madhu Sudan. Decoding of Reed-Solomon codes beyond the error-correction bound. *Journal of Complexity*, 13(1):180–193, 1997. 4.1.1

[SW19] Alexander A. Sherstov and Pei Wu. Optimal interactive coding for insertions, deletions, and substitutions. *IEEE Transactions on Information Theory*, 65(10):5971–6000, 2019. 1.4, 7, 7.1, 7.1.1, 7.1.1, 7.1.1, 7.2, 8.1.5

[SZ99] Leonard J. Schulman and David Zuckerman. Asymptotically good codes correcting insertions, deletions, and transpositions. *IEEE Transactions on Information Theory*, 45(7):2552–2557, 1999. 1.2, 1.4, 3.1, 3.1.5, 5.1.1, 5.1.3, 8, 8.1.3, 9.1.2

[Ten84] Grigory Tenengolts. Nonbinary codes, correcting single deletion or insertion (corresp.). *IEEE Transactions on Information Theory*, 30(5):766–769, 1984. 1.2

[Thu06] Axel Thue. Über unendliche zeichenreihen. *Selsk. Skr. Mat. Nat. Kl*, 7:1–22, 1906. English translation in *Selected Mathematical Papers of Axel Thue*, Universitetsforlaget, 1977. 10.1.1, 10.1.2, 10.6, 10.6.2, 10.6.2

[Thu12] Axel Thue. *Über die gegenseitige Lage gleicher Teile gewisser Zeichenreihen*. J. Dybwad, 1912. 3.4.1, 10.1.1, 10.6.1

[TV91] Michael Tsfasman and Serge G Vladut. *Algebraic-geometric codes*, volume 58. Springer Science & Business Media, 1991. 3.1.5, 3.3.1

[Ukk85] Esko Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64(1-3):100–118, 1985. 9.1.3

[Woz58] John M. Wozencraft. List decoding. *Quarterly Progress Report, Research Laboratory of Electronics, MIT*, 48:90–95, 1958. 4.1.1

[WZ18] Antonia Wachter-Zeh. List decoding of insertions and deletions. *IEEE Transactions on Information Theory*, 64(9):6297–6304, 2018. 4.1.1, 5.1.1, 5.1.4

[YKGR+15] SM Hossein Tabatabaei Yazdi, Han Mao Kiah, Eva Garcia-Ruiz, Jian Ma, Huimin Zhao, and Olgica Milenkovic. DNA-based storage: Trends and methods. *IEEE Transactions on Molecular, Biological and Multi-Scale Communications*, 1(3):230–248, 2015. 1

[Zol15] Boris Zolotov. Another solution to the Thue problem of non-repeating words. *arXiv preprint arXiv:1505.00019*, 2015. 10.1.1, 10.1.2, 10.1.2, 10.6, 10.6.1