Synchronization Strings: Codes for Insertions and Deletions Approaching the Singleton Bound*

Bernhard Haeupler
Carnegie Mellon University
Pittsburgh, PA, USA
haeupler@cs.cmu.edu

Amirbehshad Shahrasbi
Carnegie Mellon University
Pittsburgh, PA, USA
shahrasbi@cs.cmu.edu

ABSTRACT

We introduce *synchronization strings*, which provide **a novel way to efficiently deal with *synchronization errors***, i.e., insertions and deletions. Synchronization errors are strictly more general and much harder to cope with than more commonly considered *half-errors*, i.e., symbol corruptions 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 $(1+\varepsilon)k$ half-errors**. This powerful new technique has many applications. In this paper, we focus on designing *insdel codes*, i.e., error correcting block codes (ECCs) for insertion-deletion channels.

While ECCs for both half-errors and synchronization errors have been intensely studied, the later has largely resisted progress. As Mitzenmacher puts it in his 2009 survey: "*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 regime these codes are 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 straight forward 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 insdel codes achieving a rate of $1-\delta-\varepsilon$ over a constant size alphabet that efficiently correct a δ fraction of insertions or deletions.

CCS CONCEPTS

• **Mathematics of computing → Coding theory**;

KEYWORDS

Coding for Insertions and Deletions, Synchronization

ACM Reference format:
Bernhard Haeupler and Amirbehshad Shahrasbi. 2017. Synchronization Strings: Codes for Insertions and Deletions Approaching the Singleton Bound. In *Proceedings of 49th Annual ACM SIGACT Symposium on the Theory of Computing, Montreal, Canada, June 2017 (STOC'17),* 14 pages.
DOI: 10.1145/3055399.3055498

1 INTRODUCTION

Since the fundamental works of Shannon, Hamming, and others the field of coding theory has advanced our understanding of how to efficiently correct symbol corruptions and erasures. 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 problem of coding for timing errors such as closely related insertion and deletion errors, however, while also studied intensely since the 60s, has largely resisted such progress and impact so far. An expert panel [8] 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 [25] 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 [26]: *"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."* 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

*Supported in part by the National Science Foundation through grants CCF-1527110 and CCF-1618280.
A full version of this article is available at [18].

STOC'17, Montreal, Canada

DOI: 10.1145/3055399.3055498

other noise is handled more efficiently using coding techniques. We are convinced that a better theoretical understanding together with practical code constructions will eventually lead to systems which naturally and more efficiently use coding techniques to address synchronization and noise issues jointly. In addition, we feel that better understanding the combinatorial structure underlying (codes for) insertions and deletions will have impact on other parts of mathematics and theoretical computer science.

This paper 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 already found applications to channel simulations, synchronization sequences [25], interactive coding schemes [4–7, 16, 21], edit distance tree codes [2], and error correcting codes for insertion and deletions and suspect there will be many more. This paper 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 lacks behind what is known for codes for Hamming errors. While Levenshtein [22] introduced and pushed the study of such codes already in the 60s it took until 1999 for Schulman and Zuckerman [29] to construct the first insdel codes with constant rate, constant distance, and constant alphabet size. Very recent work of Guruswami et al. [10, 14] in 2015 and 2016 gave the first constant rate insdel codes for asymptotically large noise rates, via list decoding. These codes are however still polynomially far from optimal in their rate or decodable distance respectively. In particular, they achieve a rate of $\Omega(\varepsilon^5)$ for a relative distance of $1-\varepsilon$ or a relative distance of $O(\varepsilon^2)$ for a rate of $1-\varepsilon$, for asymptotically small $\varepsilon > 0$ (see Section 1.5 for a more detailed discussion of related work).

This paper 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 $0 \le \delta < 1$ and any constant $\varepsilon > 0$, there is an efficient insdel code over a constant size alphabet with block length n and rate $1-\delta-\varepsilon$ which can be uniquely and efficiently decoded from any δn insertions and deletions. The code construction takes polynomial time; and encoding and decoding can be done in linear and quadratic time, respectively. More formally, let us define 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 1.1. *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 $EditDistance(E(m), x) \le \delta n$ then $D(x) = m$. Further $\frac{k}{n} > 1-\delta-\varepsilon$, $|\Sigma| = f(\varepsilon)$, and E and D 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 straight forward application of our new synchronization strings based indexing method, is so simple that it can be summarized in one sentence:

For *any* efficient length n 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 changed by at most ε.

Far beyond just implying Theorem 1.1, this allows to instantly transfer much of the highly developed understanding for regular ECCs into the realm of insdel codes.

Theorem 1.1 is obtained by using the "near-MDS" expander codes of Guruswami and Indyk [9] as a base ECC. These codes generalize the linear time codes of Spielman [31] 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 [9] 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 usually not run in sub-quadratic time, which is not surprising given the recent SETH-conditional lower bounds [1]. For the settings of insertion-only and deletion-only errors we furthermore achieve analogs of Theorem 1.1 with linear decoding complexities (see [18]).

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 paper.

1.1.1 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, half-errors and synchronization errors. *Half-errors* consist of *erasures*, that is, a symbol being replaced with a special "?" symbol indicating the erasure, and *symbol corruptions* in which a symbol is replaced with any other symbol in Σ. The wording half-error comes from the realization that when it comes to code distances erasures are half as bad as symbol corruptions. An erasure is thus counted as one half-error while a symbol corruption counts as two half-errors (see Section 2 for more details). *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 corruption, 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 synch". 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.

1.1.2 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 stream element. Now, if we deal with only 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 incorrectly received index position caused by a deletion of an indexed symbol which is then replaced by a different symbol with the same index. This however requires two insdel errors. Generally this *trivial indexing strategy* can 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[1] 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 $\frac{Rn}{\log|\Sigma|}$ 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 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 decoding procedures.

Attaching our synchronization strings to the codewords of any efficient error correcting code, which efficiently tolerates the usual symbol corruptions and erasures, transforms any such code into an efficiently decodable insdel code while only requiring a negligible increasing in the alphabet size. This allows to use 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.

[1]Throughout this paper all logarithms are binary.

1.2 Synchronization Strings: Definition, Construction, and Decoding

Next, we want to briefly motivate and explain how we arrive at a natural definition of these magical indexing sequences S over a finite alphabet Σ and what intuition lies behind their efficient constructions and decoding procedures.

Suppose a sender has attached some indexing sequence S one-by-one to each element in a stream and consider a time t at which a receiver has received a corrupted sequence of the first t index descriptors, i.e., a corrupted version of the length t prefix of S. When the receiver tries to guess or decode the current index it should naturally consider all indexing symbols received so far and find the "best" prefix of S. This suggests that the prefix of length l of a synchronization string S acts as a codeword for the index position l and that 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 indexing symbol will lead to an unavoidable decoding error. Given this natural and unavoidable sensitivity of synchronization codes to recent corruptions, 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, to our knowledge, new 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 [2], turns out to induce a metric space on any set of strings.

With this 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 synchronization strings it became 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 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 one into the other, where $\varepsilon \geq 0$. A one line calculation also 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 set of synchronization codewords is linear and not exponential in n and (edit) distance computations (to each codeword individually) can be done via the classical Wagner-Fischer dynamic programming approach.

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 [10], which worked only for sufficiently small δ, it is a far less strong result than the near-MDS codes we promised in Theorem 1.1 for every $\delta \in (0,1)$.

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 codes, the indexing approach itself, or the encoding scheme but solely required a very different decoding strategy. In particular, instead of decoding indices in a streaming manner we consider more global decoding algorithms. We provide several such decoding algorithms in Section 6. In particular, we give a simple global decoding algorithm which for which the number of misdecodings goes to zero as the quality ε of the ε-synchronization string used goes to zero, irrespectively of how many insdel errors are applied.

Our global decoding 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 a 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 a $n^{-O(1)}$-approximate $O\left(\frac{\log n}{\log(1/\varepsilon)}\right)$-wise independent random strings satisfy the desired ε-self-matching property which, using the celebrated small sample space constructions from [28] also leads to a deterministic polynomial time construction.

1.4 Organization of this Paper

The organization of this paper closely follows the flow of the high-level description above. We start by giving more details on related work in Section 1.5 and introduce notation used in the paper in Section 2. In Section 3, we formalize the indexing problem. Section 4 shows how any solution to the indexing problem can be used to transform any regular error correcting codes into an insdel code. Section 5 introduces the relative suffix distance and ε-synchronization strings, proves the existence of ε-synchronization strings and provides an efficient construction. Section 5.2 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 6.1, introduce an efficient deterministic construction of ε-self matching strings in Section 6.2, and provide our improved decoding algorithms in the remainder of Section 6.

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 $H_q(x) = x\log(q-1) - \frac{x\log x-(1-x)\log(1-x)}{\log q}$ 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 lead 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 for a δ fraction of half-errors [33].

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 [24], which do not require to fix a desired distance at the time of encoding, explicit expander codes [9, 31] which allow linear time encoding and decoding, polar codes [13, 15] which can approach Shannon's capacity polynomially fast, network codes [23] which allow intermediate nodes in a network to recombine codewords, and efficiently list decodable codes [12] which allow to list-decode codes of relative distance δ up to a fraction of about δ symbol corruptions.

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 [30] on single-deletion codes, the 2009 survey by Mitzenmacher [26] on codes for random deletions and the most general 2010 survey by Mercier et al. [25] for the extensive work done around codes for synchronization errors and only mention the results most closely related to Theorem 1.1 here: Insdel codes were first considered by Levenshtein [22] 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 exists insdel codes capable of correcting a fraction δ of insdel erros with rate $1-\delta$, 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 [29]. Over the last two years Guruswami et al. provided new codes improving over this state of the art the asymptotically small or large noise regime 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, [14] 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 [10]. In particular, these codes achieve a rate of $\Omega(\delta^5)$ and $1-\tilde{O}(\sqrt{\delta})$ while being able to efficiently recover from a δ fraction of insertions and deletions. These works put the current state of the art for error correcting codes for insertions and deletions pretty much equal to what was known for regular error correcting codes 50 years ago, after Forney's 1965 doctoral thesis.

2 DEFINITIONS AND PRELIMINARIES

In this section, we provide the notation and definitions we will use throughout the rest of the paper.

2.1 String Notation and Edit Distance

String Notation. For two strings $S \in \Sigma^n$ and $S' \in \Sigma^{n'}$ be two strings over alphabet Σ. We define $S \cdot S' \in \Sigma^{n+n'}$ to be their concatenation. For any positive integer k we define S^k to equal k copies of S concatenated together. For $i, j \in \{1, \ldots, n\}$, we denote the substring of S from the i^{th} index through and including the j^{th} index as $S[i, j]$. Such a consecutive substring is also called a *factor* of S. For $i < 1$ we define $S[i, j] = \perp^{-i+1} \cdot S[1, j]$ where $\perp$ is a special symbol not contained in Σ. We refer to the substring from the i^{th} index through, but not including, the j^{th} index as $S[i, j)$. The substrings $S(i, j]$ and $S[i, j]$ are similarly defined. Finally, $S[i]$ denotes the i^{th} symbol of S and $|S| = n$ is the length of S. Occasionally, the alphabets we use are the cross-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 $a_i \in \Sigma_i$.

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

Definition 2.1 (Edit distance). The *edit distance* $ED(c, c')$ between two strings $c, c' \in \Sigma^*$ is the minimum number of insertions and deletions required to transform c into c'.

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, $ED(c, c') = |c| + |c'| - 2 \cdot LCS(c, c')$, where $LCS(c, c')$ is the longest common substring of c and c'.

We also use some *string matching* notation from [2]:

Definition 2.2 (String matching). 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* $ED(c, c')$ between strings $c, c, \in \Sigma^*$ is exactly equal to $\min_{\tau : c \to c'} \{sc(\tau_1) + sc(\tau_2)\}$.

3 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 indices 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. Interesting solution to the indexing problem, however, do 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.

To set up an (n, δ)-indexing problem, we fix n, i.e., the number of symbols which are being sent, and the maximum fraction δ of symbols that can be inserted or deleted. We further call the string S the *synchronization string*. Lastly, we describe the influences 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 $del(\tau_1) = S$, $del(\tau_2) = S_\tau$, 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 $i = |del(\tau_1[1, k])|$ and $j = |del(\tau_2[1, k])|$.

Definition 3.1 ((n, δ)-Indexing Algorithm). The pair $(S, \mathcal{D}_S)$ consisting of a synchronization string $S \in \Sigma^n$ and an algorithm $\mathcal{D}_S$ is called a (n, δ)-indexing algorithm over alphabet Σ if for any set of $n\delta$ insertions and deletions represented by τ which alter S to a string S_τ, the algorithm $\mathcal{D}_S(S_\tau)$ outputs either $\perp$ or an index between 1 and n for every symbol in S_τ.

The $\perp$ symbol here represents an "I don't know" response of the algorithm while an index j output by $\mathcal{D}_S(S_\tau)$ for the i^{th} symbol of S_τ should be interpreted as the (n, δ)-indexing algorithm guessing that this was the j^{th} symbol of S. One seeks algorithms that decode as many indices as possible correctly. Naturally, one can only *correctly decode* indices that were *correctly transmitted*. Next we give formal definitions of both notions:

Definition 3.2 (Correctly Decoded Index). An (n, δ) indexing algorithm $(S, \mathcal{D}_S)$ decodes index j correctly under τ 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 $\tau_2[k] = S_\tau[j]$.

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

Definition 3.3 (Successfully Transmitted Symbol). For string S_τ, which was derived from a synchronization string S via $\tau = (\tau_1, \tau_2)$, we call the j^{th} symbol $S_\tau[j]$ successfully transmitted if it stems from a symbol coming from S, i.e., if there exists a k such that $|del(\tau_2[1,k])| = j$ and $\tau_1[k] = \tau_2[k]$.

We now define the quality of an (n,δ)-indexing algorithm by counting the maximum number of misdecoded indices 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 quality definition as capturing how far from this ideal solution an algorithm is (stemming likely from the smaller alphabet which is used for S).

Definition 3.4 (Misdecodings of an (n,δ)-Indexing Algorithm). Let $(S, \mathcal{D}_S)$ be an (n,δ)-indexing algorithm. We say this algorithm has at most k misdecodings if for any τ corresponding to at most $n\delta$ insertions and deletions the number of correctly transmitted indices that are incorrectly decoded is at most k.

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

Definition 3.5 (Error-free Solution). We call $(S, \mathcal{D}_S)$ an error-free (n,δ)-indexing algorithm with respect to a set of deletion or insertion patterns if every index output is either $\perp$ or correctly decoded. In particular, the algorithm never outputs an incorrect index, even for indices which are not correctly transmitted.

It is noteworthy that error-free solutions are essentially only obtainable when dealing with the insertion-only or deletion-only setting. In both cases, the trivial solution with $S = 1,\cdots,n$ which decodes any index that was received exactly once is error-free. We later give some algorithms which preserve this nice property, even over a smaller alphabet, and show how error-freeness can be useful in the context of error correcting codes.

Lastly, another very useful property of some (n,δ)-indexing algorithms is that their decoding process operates in a streaming manner, i.e, the decoding algorithm decides the index output for $S_\tau[j]$ 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 paper, 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 3.6 (Streaming Solutions). We call $(S, \mathcal{D}_S)$ a streaming solution if the decoded index for the ith element of the received string S_τ only depends on $S_\tau[1,i]$.

Again, the trivial solution for (n,δ)-index decoding 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 an index. 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 algorithm are (a) its alphabet size $|\Sigma|$, (b) the bound on the number of misdecodings, (c) the complexity of the decoding algorithm $\mathcal{D}$, (d) the preprocessing complexity of constructing the string S, (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-freeness property.

4 INSDEL CODES VIA INDEXING

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

To this end, we simply attach S symbol-by-symbol to every codeword of C. On the decoding end, we first decode the indices of the symbols arrived using the indexing part of each received symbol and then interpret the message parts as if they have arrived in the decoded order. Indices where zero or multiple symbols are received get considered as erased. We will refer to this procedure as *the indexing procedure*. Finally, the decoding algorithm $\mathcal{D}_C$ for C is used.

Theorem 4.1. *If $(S, \mathcal{D}_S)$ guarantees k misdecodings for the (n,δ)-index problem, then the indexing procedure recovers the codeword sent up to $n\delta + 2k$ half-errors, i.e., half-error distance of the sent codeword and the one recovered by the indexing procedure is at most $n\delta + 2k$. If $(S, \mathcal{D}_S)$ is error-free, the indexing procedure recovers the codeword sent up to $n\delta + k$ half-errors.*

Proof. Consider a set insertions and deletions described by τ consisting of D_τ deletions and I_τ insertions. Note that among n encoded symbols, at most D_τ were deleted and less than k of are decoded incorrectly. Therefore, at least $n - D_\tau - k$ indices are decoded correctly. On the other hand at most $D_\tau + k$ of the symbols sent are not decoded correctly. Therefore, if the output only consisted of correctly decoded indices for successfully transmitted symbols, the output would have contained up to $D_\tau + k$ erasures and no symbol corruption, 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 original $I_\tau + k$ erasures into a corruption error by making the indexing procedure mistakenly decode an index. Overall, this can increase the number of half-errors by at most $I_\tau + k$ for a total of at most $D_\tau + k + I_\tau + k = D_\tau + I_\tau + 2k = n\delta + 2k$ half-errors. For error-free indexing algorithms, any misdecoding does not result in an incorrect index and 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 C which is resilient to $n\delta + 2k$ half-errors enables the receiver side to fully recover m.

Next, we formally state how a good (n,δ)-indexing algorithm $(S, \mathcal{D}_S)$ over alphabet Σ_S allows one to transform any regular ECC C with block length n over alphabet Σ_C which can efficiently correct half-errors, i.e., symbol corruptions and erasures, into a good insdel code over alphabet $\Sigma = \Sigma_C \times \Sigma_S$. The following Theorem is a corollary of Theorem 4.1 and the definition of the indexing procedure:

Theorem 4.2. *Given an (efficient) (n, δ)-indexing algorithm $(S, \mathcal{D}_S)$ over alphabet Σ_S with at most k misdecodings, and decoding complexity $T_{\mathcal{D}_S}(n)$ and an (efficient) ECC C over alphabet Σ_C with rate R_C, encoding complexity $T_{\mathcal{E}_C}$, and decoding complexity $T_{\mathcal{D}_C}$ 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 $R_C \cdot \left(1 - \frac{\log \Sigma_S}{\log \Sigma_C}\right)$. The encoding complexity remains $T_{\mathcal{E}_C}$, the decoding complexity is $T_{\mathcal{D}_C} + T_{\mathcal{D}_S}(n)$ and the preprocessing complexity of constructing the code is the complexity of constructing C and S. Furthermore, if $(S, \mathcal{D}_S)$ is error-free, then choosing a C which can recover only from $n\delta + k$ erasures is sufficient to produce the same quality code.*

Note that if one chooses Σ_C such that $\frac{\log \Sigma_S}{\log \Sigma_C} = o(\delta)$, the rate loss due to the attached symbols will be negligible. With all this in place one can obtain Theorem 1.1 as a consequence of Theorem 4.2.

Proof of Theorem 1.1. Given the δ and ε from the statement of Theorem 1.1 we choose $\varepsilon' = O((\varepsilon/6)^2)$ and use Theorem 6.13 to construct a string S of length n over alphabet Σ_S of size $\varepsilon^{-O(1)}$ with the ε'-self-matching property. We then use the (n, δ)-indexing algorithm $(S, \mathcal{D}_S)$ where given in Section 6.3 which guarantees that it has at most $\sqrt{\varepsilon'} = \frac{\varepsilon}{3}$ misdecodings. Finally, we choose a near-MDS expander code [9] C which can efficiently correct up to $\delta_C = \delta + \frac{\varepsilon}{3}$ half-errors and has a rate of $R_C > 1 - \delta_C - \frac{\varepsilon}{3}$ over an alphabet $\Sigma_C = \exp(\varepsilon^{-O(1)})$ such that $\log |\Sigma_C| \geq \frac{3 \log |\Sigma_S|}{\varepsilon}$. This ensures that the final rate is indeed at least $R_C - \frac{\log \Sigma_S}{\log \Sigma_C} = 1 - \delta - 3\frac{\varepsilon}{3}$ and the number of insdel errors that can be efficiently corrected is $\delta_C - 2\frac{\varepsilon}{3} \geq \delta$. The encoding and decoding complexities are furthermore straight forward and as is the polynomial time preprocessing time given Theorem 6.13 and [9]. □

5 SYNCHRONIZATION STRINGS

In this section, we formally define and develop ε-synchronization strings, which can be used as our base synchronization string S in our (n, δ)-indexing algorithms.

As explained in Section 1.2 it makes sense to think of the prefixes $S[1, l]$ of a synchronization string S as *codewords* encoding their length l, as the prefix $S[1, l]$, or a corrupted version of it, will be exactly all the indexing information that has been received by the time the l^{th} symbol is communicated:

Definition 5.1 (Codewords Associated with a Synchronization String). Given any synchronization string S 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 a synchronization string S and its associated set of codewords is:

Definition 5.2 (Relative Suffix Distance). For any two strings $S, S' \in \Sigma^*$ we define their relative suffix distance *RSD* as follows:

$$RSD(S, S') = \max_{k>0} \frac{ED\left(S(|S| - k, |S|], S'(|S'| - k, |S'|]\right)}{2k}$$

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 [2], 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.

Lemma 5.3. *For any strings S_1, S_2, S_3 we have*

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

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 $RSD(S_1, S_2) = 0$ if and only if for all k the edit distance of the k prefix of S_1 and S_2 is zero, i.e., if for every k the k-prefix 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 $\delta_1 = RSD(S_1, S_2)$ and $\delta_2 = RSD(S_2, S_3)$. By the definition of RSD this implies that for all k the k-prefixes of S_1 and S_2 have edit distance at most $2\delta_1 k$ and the k-prefixes 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-prefix of S_1 and S_3 have edit distance at most $(\delta_1 + \delta_2) \cdot 2k$ which implies that $RSD(S_1, S_3) \leq \delta_1 + \delta_2$. □

With these definitions in place, it remains to find synchronization strings whose prefixes induce a set of codewords, i.e., prefixes, with large RSD distance. It is easy to see that the RSD distance for any two strings ending on a different symbol is one. This makes the trivial synchronization string, which uses each symbol in Σ only once, induce an associated set of codewords of optimal minimum-RSD-distance one. Such trivial synchronization strings, however, are not interesting as they require an alphabet size linear in the length n. To find good synchronization 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 synchronization string, i.e., a string of n distinct symbols.

Definition 5.4 (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 $ED(S[i, j), S[j, k)) > (1 - \varepsilon)(k - i)$. We call the set of prefixes of such a string an ε-synchronization string.

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

Lemma 5.5. *If S is an ε-synchronization string, then $RSD(S[1, i], S[1, j]) > 1 - \varepsilon$ for any $i < j$, i.e., any two codewords associated with S have RSD distance of at least $1 - \varepsilon$.*

Proof. Let $k = j - i$. The ε-synchronization string property of S guarantees that $ED(S[i - k, i), S[i, j)) > (1 - \varepsilon)2k$. Note that this holds even if $i - k < 1$. To finish the proof we note that the maximum in the definition of RSD includes the term $\frac{ED(S[i-k,i), S[i,j))}{2k} > 1 - \varepsilon$, which implies that $RSD(S[1, i], S[1, j]) > 1 - \varepsilon$. □

5.1 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 the 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 [32] 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.

Theorem 5.6. *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 concatenating two strings T and R, where T is simply the repetition of $0, \ldots, t-1$ for $t = \Theta(1/\varepsilon^2)$, and R is a uniformly random string of length n over alphabet Σ. In particular, $S_i = (i \bmod t, R_i)$.

We prove that S is an ε-synchronization string by showing that there is a positive probability that S contains no *bad triple*, where (x, y, z) is a bad triple if $ED(S[x, y), S[y, z)) \leq (1-\varepsilon)(z-x)$.

First, note that a triple (x, y, z) for which $z - x < t$ 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 $ED(R[x, y), R[y, z)) = (y-x) + (z-y) - 2k = z - x - 2k$. 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 upper bound the probability that an interval of length l is bad:

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

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 valid is $|\Sigma|^{-\frac{\varepsilon l}{2}}$. The second inequality comes from the fact that $\binom{n}{k} < (ne/k)^k$.

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.

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. 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 $D > 1$. This means that it suffices to find a constant D that for all substrings $R[p, q)$ satisfies $C^{\varepsilon(p-q)} \leq D^{\varepsilon(p-q)} \prod_{l=t}^{n} (1 - D^{-\varepsilon l})^{l+(q-p)}$, or more clearly, for all $l' \in \{1, \cdots, n\}$,

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

For $D > 1$, the right-hand side of Equation (1) is maximized when $n = \infty$ and $l' = 1$, and since we want Equation (1) to hold for all n and all $l' \in \{1, \cdots, n\}$, it suffices to find a D such that $C \geq D/\prod_{l=t}^{\infty}(1 - D^{-\varepsilon l})^{\frac{l+1}{\varepsilon}}$.

To this end, let $L = \min_{D>1} \left\{ D \Big/ \prod_{l=t}^{\infty}(1 - D^{-\varepsilon l})^{\frac{l+1}{\varepsilon}} \right\}$. Then, it suffices to have $|\Sigma|$ large enough so that $C = \varepsilon\sqrt{|\Sigma|}/e \geq L$, which means that $|\Sigma| \geq \frac{e^2 L^2}{\varepsilon^2}$ suffices to allow us to use the Lovász local lemma. We claim that $L = \Theta(1)$, which will complete the proof. Since $t = \omega\left(\frac{\log(1/\varepsilon)}{\varepsilon}\right)$, for all $l \geq t$, $D^{-\varepsilon l} \cdot \frac{l+1}{\varepsilon} \ll 1$. Therefore, we can use the fact that $(1-x)^k > 1 - xk$ to show that:

$$\frac{D}{\prod_{l=t}^{\infty}\left(1 - D^{-\varepsilon l}\right)^{\frac{l+1}{\varepsilon}}} < \frac{D}{\prod_{l=t}^{\infty}\left(1 - (l+1)/\varepsilon \cdot D^{-\varepsilon l}\right)} \tag{2}$$

$$< \frac{D}{1 - \sum_{l=t}^{\infty} \frac{l+1}{\varepsilon} \cdot D^{-\varepsilon l}} \tag{3}$$

$$= D \Big/ \left[1 - \frac{1}{\varepsilon} \frac{2t\,(D^{-\varepsilon})^t}{(1 - D^{-\varepsilon})^2}\right] \tag{4}$$

$$= D \Big/ \left[1 - \frac{2}{\varepsilon^3} \frac{D^{-\frac{1}{\varepsilon}}}{(1 - D^{-\varepsilon})^2}\right]. \tag{5}$$

Equation (3) is derived using the fact that $\prod_{i=1}^{\infty}(1 - x_i) \geq 1 - \sum_{i=1}^{\infty} x_i$ and Equation (4) is a result of the following equality for $x < 1$: $\sum_{l=t}^{\infty}(l+1)x^l = x^t(1 + t - tx)/(1-x)^2 < 2tx^t/(1-x)^2$.

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) is legal and (5) can be upper-bounded by a constant. Hence, $L = \Theta(1)$ and the proof is complete. □

Remarks on the alphabet size: Theorem 5.6 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 consecutive substring of size ε^{-1} 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, $ED\left(S[j], S[j+1, j'+1))\right) = j' - j - 1 = (j' + 1 - j) - 2 \leq (j' + 1 - j)(1 - 2\varepsilon)$. We believe that using 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 improved 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 5.6 also implies an efficient randomized construction.

Lemma 5.7. *There exists a randomized algorithm which for any $\varepsilon > 0$ constructs a ε-synchronization string of length n over an alphabet of size $O(\varepsilon^{-4})$ in expected time $O(n^5)$.*

Proof. Using the algorithmic framework for the Lovász local lemma given by Moser and Tardos [27] and the extensions by Haeupler et al.[17] one can get such a randomized algorithm from the proof in Theorem 5.6. The algorithm starts with a random string over any alphabet Σ of size ε^{-C} 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 classical 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. [17] shows that this algorithm performs only $O(n)$ expected number of re-samplings. This gives an expected running time of $O(n^5)$ overall. □

We remark that any string produced by the randomized construction of Lemma 5.7 is guaranteed to be a correct ε-synchronization string (not just with probability one). 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 6 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.

5.2 Decoding

We now provide an algorithm for decoding 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 5, we introduced the notion of relative suffix distance between two strings. Theorem 5.5 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 decoding scheme for detecting the index of a received symbol would be finding the prefix 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 i* which presents the maximized density of errors taken place over suffixes of $S[1, i]$. We will introduce a very natural decoding approach for synchronization strings that simply works by decoding a received string by finding the codeword of a synchronization string S (prefix of synchronization string) with minimum distance to the received string. 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 many 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 decoding scheme decodes indices of received symbols correctly for all but $\frac{2c}{1-\varepsilon}$ many of successfully transmitted symbols. Formally, we claim that:

Theorem 5.8. *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 decoding algorithm is streaming and can be implemented so that it works in $O(n^4)$ time.*

Before proceeding to the formal statement and the proofs of the claims above, we first provide the following useful definitions.

Definition 5.9 (Error Count Function). Let S be a 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. $\mathcal{E}(i, j)$ counts the potential deletion of $S[j]$. However, it does *not* count the potential deletion of $S[i]$.

Definition 5.10 (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 $\max_{i \geq 1} \frac{\mathcal{E}(|S|-i,|S|)}{i}$.

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

Lemma 5.11. *Let string S be sent over an insertion-deletion channel and the corrupted message S' be received on the other end. The relative suffix distance $RSD(S, S')$ between the string S that was sent and the string S' which was received 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:

$$RSD(S, S') = \max_{k>0} ED(S(|S| - k, |S|], S'(|S'| - k, |S'|])/2k \quad (6)$$

$$= \max_{k>0} \frac{\min_{\tau: S(|S|-k,|S|] \to S'(|S'|-k,|S'|]} sc(\tau_1) + sc(\tau_2)}{2k} \quad (7)$$

$$\leq \max_{k>0}(sc(\tau'_1) + sc(\tau'_2))/k \leq \text{Rel. Suff. Error Dens.} \quad (8)$$

where τ' is $\tilde{\tau}$ limited to its suffix corresponding to $S(|S| - k, |S|])$. Note that Steps (6) and (7) follow from the definitions of edit distance and relative suffix distance. Moreover, to see Step (8), one has to note that one single insertion or deletion on the k-element suffix of a string may result into 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 5.11 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 corruptions 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 the corrupted codeword he received by simply finding the codeword with minimum relative suffix distance.

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

Lemma 5.12. *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 $\Phi(i) = \max_{1\le s\le i}\left\{\frac{t\cdot\mathcal{E}(i-s,i)}{1-\varepsilon} - s\right\}$. 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 at most $\frac{tk}{1-\varepsilon} - 1$, i.e., $\Phi(i) \le \Phi(i-1) + \frac{tk}{1-\varepsilon} - 1$.
(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 $\frac{t\cdot(c_i+c_d)}{1-\varepsilon}$. Therefore, the value of Φ can be non-zero for at most $\frac{t\cdot(c_i+c_d)}{1-\varepsilon}$ many inputs. As value of $\Phi(i)$ is non-zero for all i's where $S[i]$ has been removed by adversary, there are at most $\frac{t\cdot(c_i+c_d)}{1-\varepsilon} - c_d$ indices 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 be decoded incorrectly.

We now proceed to the proof of above claims to finish the proof:

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

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

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

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

(3) As $\Phi(i) = 0$, for all $1 \le s \le i$, $\frac{t\cdot\mathcal{E}(i-s,i)}{1-\varepsilon} - s \le 0$. Therefore, for all $1 \le s \le i$, $t\cdot\mathcal{E}(i-s, i) \le s(1-\varepsilon)$. This gives that Relative Suffix Error Density $= \max_{1\le s\le i}\{\mathcal{E}(i-s,i)/s\} \le (1-\varepsilon)/t$.

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 5.8. As adversary is allowed to insert or delete up $n\delta$ symbols, by Lemma 5.12, 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 5.11, there are at most $\frac{2n\delta}{1-\varepsilon}$ misdecoded successfully transmitted symbols.

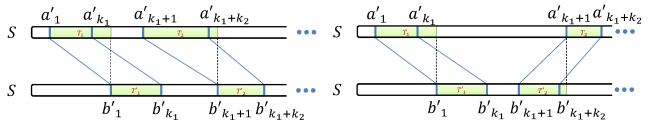
Further, we remark that this algorithm can be implemented in $O(n^4)$ as follows: Using dynamic programming, we can pre-process the edit distance of any consecutive substring of S, like $S[i, j]$ to any consecutive substring of S', like $S'[i', j']$, in $O(n^4)$. Then, for each symbol of the received string, like $S'[l']$, 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 $\frac{ED(S(l-k,l],S'(l'-k,l'])}{k}$ over k which can be done in $O(n)$. With a $O(n^4)$ pre-process and a $O(n^3)$ computation as mentioned above, we have shown that the decoding process can be implemented in $O(n^4)$. □

6 MORE ADVANCED GLOBAL DECODING ALGORITHMS

Thus far, we have introduced ε-synchronization strings as fitting solutions to the indexing problem. In Section 5.2, 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 1.3, such a guarantee falls short of giving Theorem 1.1. In this section, we thus provide a more advanced decoding algorithm that achieves a misdecoding fraction which vanishes as ε goes to zero.

We start by pointing out a very useful property of ε-synchronization strings in Section 6.1. 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 6.2). 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 6.3, we propose a decoding 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 $O\left(n\sqrt{\varepsilon}\right)$



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

Figure 1: Pictorial representation of T_2 and T'_2

misdecodings. This algorithm works in time $O(n^2/\sqrt{\varepsilon})$ and is exactly what we need to prove our main theorem.

6.1 Monotone Matchings and the ε-Self Matching Property

Before proceeding to the main results of this section, we start by defining *monotone matchings* which provide a formal way to refer to common substrings of two strings:

Definition 6.1 (Monotone Matchings). A monotone matching between S and S' is a set of pairs 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 decoding algorithms. Basically, Theorem 6.2 states that two 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 6.2. *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, $b \leq \varepsilon(n-g)$.*

Proof. Let $(a'_1, a'_2), \cdots, (a'_{m'}, b'_{m'})$ indicate the set of bad pairs in M indexed as $a'_1 < \cdots < a'_{m'}$ and $b'_1 < \cdots < b'_{m'}$. 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 $(a'_1, a'_2), \cdots, (a'_{k_1}, b'_{k_1})$ form a common substring 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 $k_1 \leq LCS(T_1, T'_1) \leq \left[|T_1| + |T'_1| - ED\left(T_1, T'_1\right)\right]/2 \leq \varepsilon(|T_1| + |T'_1|)/2$.

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:

$$k_2 \leq \varepsilon(|T_2| + |T'_2|)/2 \tag{9}$$

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 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 (9),

$$\sum_{i=1}^{l} k_i \leq \frac{\varepsilon}{2} \cdot \left(\sum_{i=1}^{l} |T_i| + \sum_{i=1}^{l} |T'_i|\right) \tag{10}$$

Note that $\sum_{i=1}^{l} k_i = u$ and T_is are mutually exclusive and contain no indices where a good pair occurs at. 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 (10) give that: $u \leq \frac{\varepsilon}{2} \cdot 2\,(n-g) \Rightarrow b \leq \varepsilon(n-g)$ □

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

Definition 6.3 (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 6.4. *ε-synchronization and ε-self matching properties are related as follows: (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. Part (a) is a straightforward consequence of Theorem 6.2. To prove part (b), assume by contradiction that there are $i < j < k$ such that $ED(S[i, j), S[j, k)) \leq (1-\varepsilon)(k-i)$. Then, $LCS(S[i, j), S[j, k)) \geq \frac{k-i-(1-\varepsilon)(k-i)}{2} = \frac{\varepsilon}{2}(k-i)$. The corresponding pairs of such longest common substring form a monotone matching of size $\frac{\varepsilon}{2}(k-i)$ which contradicts $\frac{\varepsilon}{2}$-self-matching property of S. □

As a matter of fact, the decoding algorithm that we will propose for ε-synchronization strings in Section 6.3 only makes use of the ε-self-matching property of ε-synchronization strings.

We now proceed to the definition of ε-bad-indices which will enable us to show that ε-self matching property, as opposed to the ε-synchronization property, is robust against local changes.

Definition 6.5 (ε-bad-index). We call index k of string S an ε-bad-index 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 index k blames interval $[i, j]$.

Using the notion of ε-bad indices, we now present Lemma 6.6. This lemma suggests that a string containing limited fraction of ε-bad indices 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 ε-synchronization property does not satisfy any such robustness quality.

Lemma 6.6. *If the fraction of ε-bad indices 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 indices of S is at most $|S|(1-\gamma)\varepsilon$ by definition. Further, each ε-bad index can appear at most once in each end of bad pairs. Therefore, the number of bad pairs in S can be at most $|S|(1-\gamma)\varepsilon + 2|S|\gamma \leq |S|(\varepsilon + 2\gamma)$, which 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 indices for sufficiently large $\varepsilon' > \varepsilon$. The proof of Lemma 6.7 is available in the full version of this paper [18].

Lemma 6.7. *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 indices of S can be ε'-bad.*

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 for indexing together with an ε-self matching string leads to guarantees on the misdecoding performance which are 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 index in any arbitrary string in Theorem 6.8. Then, using Theorem 6.8 and Lemma 6.7, we upper-bound the number of misdecodings that may happen using a minimum RSD decoder along with an ε-self matching string in Theorem 6.9.

Lemma 6.8. *Let S be an arbitrary string of length n and $1 \le i \le n$ be such that i'th index of S is not an ε-bad index. Then, for any $j \neq i$, $RSD(S[1,i], S[1,j]) > 1-\varepsilon$.*

The proof of Lemma 6.8 is available in [18].

Theorem 6.9. *Using any ε-self matching string along with minimum RSD algorithm, one can solve the (n,δ)-indexing problem with a guarantee of $n(4\delta + 6\varepsilon)$ misdecodings.*

Proof. Note that applying Lemma 6.7 for ε' gives that there are at most $\frac{3n\varepsilon}{\varepsilon'}$ indices in S that are ε'-bad. Further, using Theorem 5.8 and Lemma 6.8, 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. □

6.2 Construction of ε-Self Matching Strings

In this section, we will use Lemma 6.6 to show that there is a polynomial deterministic construction of a string of length n with the ε-self-matching property, which can then for example be used to obtain a deterministic code construction. We start by showing that even random strings satisfy the ε-selfmatching property for an ε polynomial in the alphabet size:

Theorem 6.10. *A random string on 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 $|\Sigma| = \varepsilon^{-3}$. We are going to find the expected number of ε-bad indices in S. We first count the expected number of ε-bad indices that blame intervals of length $\frac{2}{\varepsilon}$ or smaller. If index 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 gives that there are two identical elements in $4\varepsilon^{-1}$ neighborhood of S. Therefore, the probability of index k being ε-bad blaming $S[i,j]$ for $j - i < 2\varepsilon^{-1}$ can be upper-bounded by $\binom{4\varepsilon^{-1}}{2}\frac{1}{|\Sigma|} \le 8\varepsilon$. 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 indices in S blaming intervals of length $2\varepsilon^{-1}$ or more. Let $B_{i,j}$ denote the event that $S[i,j)$ does not satisfy ε-self-matching property. Since every interval of length l which does not satisfy ε-self-matching property causes at most l ε-bad indices, we get that the expected fraction of such indices, i.e., γ', is at most:

$$\mathbb{E}[\gamma'] = \frac{1}{n}\sum_{l=2\varepsilon^{-1}}^{n}\sum_{i=1}^{n} l\cdot \Pr[B_{[i,i+l)}] \le \sum_{l=2\varepsilon^{-1}}^{n} l\binom{l}{l\varepsilon}^2\frac{1}{|\Sigma|^{l\varepsilon}} \qquad (11)$$

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 being a matching between random string S and itself is $\frac{1}{|\Sigma|^{l\varepsilon}}$. Then,

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

Note that $\sum_{l=2\varepsilon^{-1}}^{\infty} lx^l = 2\varepsilon^{-1}x^{2\varepsilon^{-1}} - (2\varepsilon^{-1}-1)x^{2\varepsilon^{-1}+1}/(1-x)^2$ 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,

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

Using Lemma 6.6, 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 $\Theta(\log n/\log(1/\varepsilon))$-wise independent [28] 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 [28]. The proof of Theorem 6.11 follows a similar strategy as was used in [3] to derandomize the constructive Lovász local lemma. In particular the crucial idea, given by Claim 6.12, 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 6.11. *A $\frac{c\log n}{\log(1/\varepsilon)}$-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 is a sufficiently large constant.*

Proof. Let S be a pseudo-random string of length n with $\frac{c\log n}{\log(1/\varepsilon)}$-wise independent symbols. Then, Step (11) 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 that:

Claim 6.12. *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 Claim 6.12, 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 subintervals 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 (11) can be revised by upper-bounding 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,

$$\begin{aligned}
\Pr[A_{i,l}] &\leq \Pr\left[S \text{ contains } I_1, I_2 : |I_i| = \frac{c\log n}{\varepsilon\log(1/\varepsilon)} \right.\\
&\qquad \left. \text{and } LCS(I_1, I_2) \geq 0.99\frac{c\log n}{2\log(1/\varepsilon)}\right]\\
&\leq n^2\binom{\varepsilon^{-1}c\log n/\log(1/\varepsilon)}{0.99c\log n/2\log(1/\varepsilon)}^2\left(\frac{1}{|\Sigma|}\right)^{\frac{c\log n}{2\log(1/\varepsilon)}}\\
&\leq n^2\left(2.04e\varepsilon^{-1}\right)^{\frac{2\times 0.99c\log n}{2\log(1/\varepsilon)}}\varepsilon^{\frac{6c\log n}{2\log(1/\varepsilon)}}\\
&= n^2(2.04e)^{\frac{0.99c\log n}{\log(1/\varepsilon)}}\varepsilon^{\frac{4.02c\log n}{2\log(1/\varepsilon)}}\\
&= n^{2+\frac{c\ln(2.04e)}{\log(1/\varepsilon)}-2.01c} < n^{2-c/4} = O\left(n^{-c'}\right)
\end{aligned}$$

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 (11) can be revised as:

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

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

Proof of Claim 6.12. Let M be a self-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 $2l/m$ intervals for which $|E_{i,j}| \neq 0$ and subsequently $|E_{i,j}|/m \neq 0$. Hence, on the right hand side we have the average of $2l/m$ many non-zero terms which is greater than $\varepsilon/2$. So, there has 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 [28] Theorem 6.11 directly leads to a deterministic algorithm for finding a string of size n with ε-self-matching property. For this one simply checks all possible points in the sample space of the $\frac{c\log n}{\log(1/\varepsilon)}$-wise independent strings and finds a string S with $\gamma_S \leq \mathbb{E}[\gamma] = O(\varepsilon)$. In other words, using brute-force, one can find a string satisfying $O(\varepsilon)$-self-matching property in $O(|\Sigma|^{c\log n/\log(1/\varepsilon)}) = n^{O(1)}$.

Theorem 6.13. *There is a deterministic algorithm running in $n^{O(1)}$ that finds a string of length n satisfying ε-self-matching property over an alphabet of size $O(\varepsilon^{-6})$.*

6.3 Global Decoding Algorithm

Now, we provide an alternative indexing algorithm to be used along with ε-synchronization strings. 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 being affected by up to $n\delta$ insertions or deletions. Furthermore, let d_i symbols be inserted into the communication and d_r symbols be deleted from it.

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' (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 algorithm, $S'[t_i]$ is decoded as $S[i]$ if and only if $S[i]$ is only matched to $S'[t_i]$ in all $M_1, \cdots, M_{1/\beta}$. Note that the longest common subsequence of two strings of length $O(n)$ can be found in $O(n^2)$ 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.

Theorem 6.14. *This decoding algorithm 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 will be $O\left(n\sqrt{\varepsilon}\right)$ and running time will be $O\left(n^2/\sqrt{\varepsilon}\right)$.*

Proof. First, we claim that at most $(n + d_i - d_r)\beta$ many 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 longest common substrings. Hence, size of any of M_is is at least $(n+d_i-d_r)\beta$. So, the summation of their sizes exceeds $(n + d_i - d_r)\beta \times \frac{1}{\beta} = |S'|$ which brings us to a contradiction.

Furthermore, as a result of Theorem 6.4, any of M_is contain at most εn many incorrectly matched elements. Hence, at least $\frac{\varepsilon}{\beta}n$ many of the matched symbols are matched to incorrect index. Therefore, the total number of misdecodings can be bounded by $(n + d_i - d_r)\beta + \frac{\varepsilon}{\beta}n$. □

7 CONCLUSION AND FUTURE WORK

This paper introduced synchronization strings as simple yet very powerful mathematical objects that are designed to deal with synchronization errors in communications. In particular, one can use them to efficiently transform insertion and deletion errors into much simpler Hamming-type errors. This, e.g., transforms regular ECCs into insdel codes with the same optimal rate/distance tradeoff.

The main focus of this paper lies in developing the theory of synchronization strings and its application to the design of essentially optimal insdel codes over large finite alphabets. However, our method can also be ported into the setting of binary codes [20]: One can fairly easily obtain binary error correcting codes that tolerate a ε fraction of synchronization errors while achieving a rate of $1 - O(\sqrt{\varepsilon \log(1/\varepsilon)})$ with a linear encoding and an $O(n^2)$ time decoding algorithm. This is improves over the current state of the art insdel codes [11] in terms of simplicity, encoding and decoding complexity (which for the codes in [11] is $n^{\varepsilon^{-O(1)}}$ or $n^{O(1)}(\log n)^{\varepsilon^{-O(1)}}$ at the cost of a quadratically worse distance), and slightly also in the rate/distance tradeoff.

In this article, we give a randomized $n^{O(1)}$ time construction for ε-synchronization strings and a $n^{O(1)}$ time deterministic construction for ε-selfmatching strings of length n. In [19] several highly parallel, deterministic linear time constructions for ε-synchronization strings are given. In fact, these constructions provide highly-explicit infinite synchronization strings ε-synchronization strings in which the i^{th} symbol can be computed deterministically in $O(\log i)$ time. We also give strengthened versions of ε-synchronization strings which come with very fast *local* decoding procedures and discuss several further applications.

In [20] it is shown that our synchronization string based indexing method can be extended to fully simulate an ordinary corruption channel over a insertion-deletion channel. This is much stronger than constructing insdel codes and allows to completely hide the existence of synchronization errors in many applications that go way beyond codes, such as, settings with feedback, real-time control, or settings with interactive communications. This directly leads to new interactive coding schemes for the setting with synchronization errors [2] including the first efficient scheme and the first scheme with good (and in fact likely near optimal) communication rate for small error fractions.

We strongly believe that synchronization strings, our indexing method, and channel simulations will lead to many further applications in the future. We also believe that ε-synchronization strings, in their own right, are a worthwhile and interesting combinatorial structure to study.

ACKNOWLEDGEMENTS

The authors want to thank Ellen Vitercik and Allison Bishop for valuable discussions in the early stages of this work.

REFERENCES

[1] Arturs Backurs and Piotr Indyk. 2015. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). *Proceedings of the ACM Symposium on Theory of Computing (STOC)* (2015), 51–58.

[2] Mark Braverman, Ran Gelles, Jieming Mao, and Rafail Ostrovsky. 2016. Coding for interactive communication correcting insertions and deletions. *Proceedings of the International Conference on Automata, Languages, and Programming (ICALP)* 55 (2016), 61:1–61:14.

[3] Karthekeyan Chandrasekaran, Navin Goyal, and Bernhard Haeupler. 2013. Deterministic algorithms for the Lovász local lemma. *SIAM J. Comput.* 42, 6 (2013), 2132–2155.

[4] Ran Gelles. 2015. Coding for Interactive Communication: A Survey. (2015).

[5] Ran Gelles and Bernhard Haeupler. 2015. Capacity of Interactive Communication over Erasure Channels and Channels with Feedback. *Proceeding of the ACM-SIAM Symposium on Discrete Algorithms (SODA)* (2015), 1296–1311.

[6] Mohsen Ghaffari and Bernhard Haeupler. 2014. Optimal Error Rates for Interactive Coding II: Efficiency and List Decoding. *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)* (2014), 394–403.

[7] Mohsen Ghaffari, Bernhard Haeupler, and Madhu Sudan. 2014. Optimal Error Rates for Interactive Coding I: Adaptivity and other Settings. *Proceedings of the ACM Symposium on Theory of Computing (STOC)* (2014), 794–803.

[8] SW Golomb, J Davey, I Reed, H Van Trees, and J Stiffler. 1963. Synchronization. *IEEE Transactions on Communications Systems* 11, 4 (1963), 481–491.

[9] Venkatesan Guruswami and Piotr Indyk. 2005. Linear-time encodable/decodable codes with near-optimal rate. *IEEE Transactions on Information Theory (TransInf)* 51, 10 (2005), 3393–3400.

[10] Venkatesan Guruswami and Ray Li. 2016. Efficiently decodable insertion/deletion codes for high-noise and high-rate regimes. *Proceedings of IEEE International Symposium on Information Theory (ISIT)* (2016), 620–624.

[11] Venkatesan Guruswami and Ray Li. 2016. Efficiently decodable insertion/deletion codes for high-noise and high-rate regimes. *arXiv:1605.04611* (2016).

[12] Venkatesan Guruswami and Atri Rudra. 2008. Explicit codes achieving list decoding capacity: Error-correction with optimal redundancy. *IEEE Transactions on Information Theory (TransInf)* 54, 1 (2008), 135–150.

[13] Venkatesan Guruswami and Ameya Velingker. 2015. An entropy sumset inequality and polynomially fast convergence to shannon capacity over all alphabets. *Proceedings of the 30th Conference on Computational Complexity* (2015), 42–57.

[14] Venkatesan Guruswami and Carol Wang. 2017. Deletion codes in the high-noise and high-rate regimes. *IEEE Transactions on Information Theory (TransInf)* 63, 4 (2017), 1961–1970.

[15] Venkatesan Guruswami and Patrick Xia. 2015. Polar codes: Speed of polarization and polynomial gap to capacity. *IEEE Transactions on Information Theory (TransInf)* 61, 1 (2015), 3–16.

[16] Bernhard Haeupler. 2014. Interactive Channel Capacity Revisited. *Proceeding of the IEEE Symposium on Foundations of Computer Science (FOCS)* (2014), 226–235.

[17] Bernhard Haeupler, Barna Saha, and Aravind Srinivasan. 2011. New Constructive Aspects of the Lovász Local Lemma. *Journal of the ACM (JACM)* 58, 6 (2011), 28.

[18] Bernhard Haeupler and Amirbehshad Shahrasbi. 2017. Synchronization Strings: Codes for Insertions and Deletions Approaching the Singleton Bound. *arXiv:1704.00807* (2017).

[19] Bernhard Haeupler and Amirbehshad Shahrasbi. 2017. Synchronization Strings: Explicit Constructions, Local Decoding and Applications. (2017).

[20] Bernhard Haeupler, Amirbehshad Shahrasbi, and Ellen Vitercik. 2017. Synchronization Strings: Channel Simulations and Interactive Coding for Insertions and Deletions. (2017).

[21] Gillat Kol and Ran Raz. 2013. Interactive channel capacity. *Proceedings of the ACM Symposium on Theory of Computing (STOC)* (2013), 715–724.

[22] Vladimir I Levenshtein. 1965. Binary codes capable of correcting deletions, insertions, and reversals. *Doklady Akademii Nauk SSSR 163* 4 (1965), 845–848.

[23] S-YR Li, Raymond W Yeung, and Ning Cai. 2003. Linear network coding. *IEEE Transactions on Information Theory (TransInf)* 49, 2 (2003), 371–381.

[24] Michael Luby. 2002. LT codes. *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)* (2002), 271–282.

[25] Hugues Mercier, Vijay K Bhargava, and Vahid Tarokh. 2010. A survey of error-correcting codes for channels with symbol synchronization errors. *IEEE Communications Surveys & Tutorials* 1, 12 (2010), 87–96.

[26] Michael Mitzenmacher. 2009. A survey of results for deletion channels and related synchronization channels. *Probability Surveys* 6 (2009), 1–33.

[27] Robin A. Moser and Gabor Tardos. 2010. A constructive proof of the general Lovász Local Lemma. *Journal of the ACM (JACM)* 57, 2 (2010), 11.

[28] Joseph Naor and Moni Naor. 1993. Small-bias probability spaces: Efficient constructions and applications. *SIAM Journal on Computing (SICOMP)* 22, 4 (1993), 838–856.

[29] Leonard J. Schulman and David Zuckerman. 1999. Asymptotically good codes correcting insertions, deletions, and transpositions. *IEEE Transactions on Information Theory (TransInf)* 45, 7 (1999), 2552–2557.

[30] Neil JA Sloane. 2002. On single-deletion-correcting codes. *Codes and Designs, de Gruyter, Berlin* (2002), 273–291.

[31] Daniel A Spielman. 1996. Linear-time Encodable and Decodable Error-Correcting Codes. *IEEE Transactions on Information Theory (TransInf)* 42, 6 (1996), 1723–1731.

[32] A Thue. 1977. Uber die gegenseitige Lage gleicher Teile gewisser Zeichenreihen (1912). *Selected mathematical papers of Axel Thue, Universitetsforlaget* (1977).

[33] Michael Tsfasman and Serge G Vladut. 2013. *Algebraic-geometric codes*. Vol. 58. Springer Science & Business Media.