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

BERNHARD HAEUPLER and AMIRBEHSHAD SHAHRASBI, Carnegie Mellon University, USA

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 *Hamming-type errors*, i.e., symbol substitutions and erasures. For every $\varepsilon > 0$, synchronization strings allow us to index a sequence with an $\varepsilon^{-O(1)}$-size alphabet, such that one can **efficiently transform k synchronization errors into $(1 + \varepsilon)k$ Hamming-type errors**. This powerful new technique has many applications. In this article, we focus on designing *insdel codes*, i.e., error correcting block codes (ECCs) for insertion-deletion channels.

While ECCs for both Hamming-type errors and synchronization errors have been intensely studied, the latter has largely resisted progress. As Mitzenmacher puts it in his 2009 survey [30]: "*Channels with synchronization errors… are simply not adequately understood by current theory. Given the near-complete knowledge, we have for channels with erasures and errors...our lack of understanding about channels with synchronization errors is truly remarkable.*" Indeed, it took until 1999 for the first insdel codes with constant rate, constant distance, and constant alphabet size to be constructed and only since 2016 are there constructions of constant rate insdel codes for asymptotically large noise rates. Even in the asymptotically large or small noise regimes, 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 straightforward application of our synchronization strings-based indexing method gives a simple black-box construction that **transforms any ECC into an equally efficient insdel code** with only a small increase in the alphabet size. This instantly transfers much of the highly developed understanding for regular ECCs into the realm of insdel codes. Most notably, for the complete noise spectrum, we obtain efficient "near-MDS" insdel codes, which get arbitrarily close to the optimal rate-distance tradeoff given by the Singleton bound. In particular, for any $\delta \in (0, 1)$ and $\varepsilon > 0$, we give a family of insdel codes achieving a rate of $1 - \delta - \varepsilon$ over a constant-size alphabet that efficiently corrects a δ fraction of insertions or deletions.

CCS Concepts: • **Mathematics of computing → Coding theory;**

Additional Key Words and Phrases: Coding for insertions and deletions, synchronization

ACM Reference format:
Bernhard Haeupler and Amirbehshad Shahrasbi. 2021. Synchronization Strings: Codes for Insertions and Deletions Approaching the Singleton Bound. *J. ACM* 68, 5, Article 36 (September 2021), 39 pages.
https://doi.org/10.1145/3468265

This work is supported in part by NSF Grants No. CCF-1527110, No. CCF-1618280, No. CCF-1814603, No. CCF-1910588, NSF CAREER Award No. CCF-1750808, a Sloan Research Fellowship, and funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (ERC Grant Agreement No. 949272).
Authors' addresses: B. Haeupler and A. Shahrasbi, Carnegie Mellon University, Computer Science Department, Pittsburgh, PA, 15203, USA; emails: {haeupler, shahrasbi}@cs.cmu.edu.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2021 Copyright held by the owner/author(s). Publication rights licensed to ACM.
0004-5411/2021/09-ART36 $15.00
https://doi.org/10.1145/3468265

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 substitutions 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. While also studied intensely since the 1960s, the problem of coding for timing errors such as closely related insertion and deletion errors, has largely resisted such progress and impact so far. An expert panel [10] 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 [29] 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 [30]: *"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 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 that 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 article introduces synchronization strings, a new combinatorial structure that 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 [29], interactive coding schemes [6–9, 18, 25], edit distance tree codes [2], and error correcting codes for insertion and deletions and suspect there will be many more. This article focuses on the last application, namely, designing efficient error correcting block codes over large alphabets for worst-case insertion-deletion channels.

The knowledge on efficient error correcting block codes for insertions and deletions, also called *insdel codes*, severely lags behind what is known for codes for Hamming errors. While Levenshtein [26] introduced and pushed the study of such codes already in the 1960s it took until 1999 for Schulman and Zuckerman [34] to construct the first insdel codes with constant rate, constant distance, and constant alphabet size. Recent works of Guruswami et al. [13, 16] 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 $\Omega(\epsilon^5)$ for a relative distance of $1-\epsilon$ or a relative distance of $O(\epsilon^2)$ for a rate of $1-\epsilon$, for asymptotically small $\epsilon > 0$ (see Section 1.5 for a more detailed discussion of related work).

This article essentially closes this line of work by designing efficient "near-MDS" insdel codes, which approach the optimal rate-distance trade-off given by the Singleton bound. We prove that for any $0 \leq \delta < 1$ and any constant $\varepsilon > 0$, there is an efficient family of insdel codes over a constant-size alphabet with rate $1 - \delta - \varepsilon$, which can be uniquely and efficiently decoded from any δ fraction of insertions and deletions. The code construction takes polynomial time; and encoding and decoding can be done in linear and quadratic time, respectively. More formally, 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.[1]

THEOREM 1.1. *For any $\varepsilon > 0$ and $\delta \in (0, 1)$ there exists an encoding map $\mathrm{Enc} : \Sigma^k \to \Sigma^n$ and a decoding map $\mathrm{Dec} : \Sigma^* \to \Sigma^k$, such that for any $m \in \Sigma^k$, if $\mathrm{EditDistance}(\mathrm{Enc}(m), x) \leq \delta n$ then $\mathrm{Dec}(x) = m$. Further, $\frac{k}{n} > 1 - \delta - \varepsilon$, $|\Sigma| = f(\varepsilon)$, and Enc and Dec are explicit and can be computed in linear and quadratic time in n.*

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

For *any* efficient ECC with alphabet bit size $\frac{\log \varepsilon^{-1}}{\varepsilon}$, attaching to every codeword, symbol by symbol, a random or suitable pseudo-random string over an alphabet of bit size $\log \varepsilon^{-1}$ results in an efficient insdel code with a rate and decodable distance that are changed by at most ε.

Far beyond just implying Theorem 1.1, this enables us 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 [12] as a base ECC. These codes generalize the linear time codes of Spielman [36] 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 Reference [12] but still pretty fast. In particular, a quadratic time decoding for an insdel code is generally very good given that, in contrast to Hamming codes, even computing the distance between the received and the sent/decoded string is an edit distance computation. Edit distance computations in general do not run in 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 time complexities.

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 article.

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, Hamming-type errors and synchronization errors.

[1]Oftentimes in the literature, the edit distance is defined as the minimum number of insertions, deletions, or substitutions required to convert one string to another. With our definition, however, a single substitution is counted as two operations—a deletion followed by an insertion at the same position.

Hamming-type errors consist of *erasures*, that is, a symbol being replaced with a special "?" symbol indicating the erasure, and *symbol substitutions* in which a symbol is replaced with any other symbol in Σ. In this article, we measure Hamming-type errors in terms of *half-errors*. The wording half-error comes from the realization that, when it comes to code distances, erasures are half as bad as symbol substitution. An erasure is thus counted as one half-error while a symbol substitution counts as two half-errors (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 substitution, worth two half-errors, can also be achieved via a deletion followed by an insertion. Any erasure can furthermore be interpreted as a deletion together with the often very helpful extra information where this deletion took place. This makes synchronization errors at least as hard as half-errors. The real problem that synchronization errors bring with them, however, is that they cause sending and receiving parties to become "out of sync." This easily changes how received symbols are interpreted and makes designing codes or other systems tolerant to synchronization errors an inherently difficult and significantly less well understood problem.

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 element of the stream. Now, if we only deal with deletions, then 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 that is received more than once as erased. If both insertions and deletions are allowed, then one might still have elements with a spoofed or substituted value caused by a deletion of the indexed symbol, which is then replaced by a different symbol with the same index inserted. This, however, requires two insdel errors. Generally, this *trivial indexing strategy* can be seen to successfully *transform any k synchronization errors into at most k half-errors*.

In many applications, however, this trivial indexing cannot be used, because having to attach a $\log n$ bit[2] long index description to each element of an n-long stream is prohibitively costly. Consider, for example, an error correcting code of constant rate R over some potentially large but nonetheless constant-size alphabet Σ, which encodes $nR\log|\Sigma|$ bits into n symbols from Σ. Increasing Σ by a factor of n to allow each symbol to carry its $\log n$ bit index would destroy the desirable property of having an alphabet that is independent from the block length n and would furthermore reduce the rate of the code from R to $\Theta(\frac{R}{\log n})$, which approaches zero for large block lengths. For streams of unknown or infinite length such problems become even more pronounced.

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

[2]Throughout this article all logarithms are binary.

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

1.2 Synchronization Strings: Definition, Construction, and Decoding

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

Suppose that a sender has attached the symbols of an index sequence S to elements of a communication stream and consider the time at which the receiver has received a corrupted sequence of the first t index descriptors, i.e., a corrupted version of t-long prefix of S. As the receiver tries to guess or *decode* the true position of the last received symbol at this time, it should naturally consider all index symbols received so far and find the “most plausible” prefix of S. This suggests that the prefix of length l of a synchronization string S acts as a codeword for the position l and one should think of the set of prefixes of S as a code associated with the synchronization string S. Naturally, one would want such a code to have good distance properties between any two codewords under some distance measure. While edit distance, i.e., the number of insertions and deletions needed to transform one string into another seems like the right notion of distance for insdel errors in general, the prefix nature of the codes under consideration will guarantee that codewords for indices l and $l' > l$ will have edit distance exactly $l' - l$. This implies that even two very long codewords only have a tiny edit distance. On the one hand, this precludes synchronization codes with a large relative edit distance between its codewords. On the other hand, one should see this phenomenon as simply capturing the fact that at any time a simple insertion of an incorrect symbol carrying the correct next index symbol will lead to an unavoidable decoding error. Given this natural and unavoidable sensitivity of synchronization codes to recent errors, it makes sense to, instead, use a distance measure that captures the recent density of errors. In this spirit, we suggest the definition of a new (to our knowledge) string distance measure, which we call *relative suffix distance*, which intuitively measures the worst fraction of insdel errors to transform suffixes, i.e., recently sent parts of two strings, into each other. This natural measure, in contrast to a similar measure defined in Reference [2], turns out to induce a metric space on any set of strings.

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

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

1.3 More Sophisticated Decoding Procedures

All this provides an indexing solution that transforms any k synchronization errors into at most $(5+\varepsilon)k$ half-errors. This already leads to insdel codes that 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 Reference [13], 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 $\delta \in (0,1)$.

We were able to slightly improve upon the above strategy by considering an alternative to the relative suffix distance measure, which we call **relative suffix pseudo-distance (RSPD)**. RSPD was introduced in Reference [2] and, while neither being symmetric nor satisfying the triangle inequality, can act as a pseudo-distance in the minimum-distance decoder. For any set of $k = k_i + k_d$ insdel errors consisting of k_i insertions and k_d deletions, this improved indexing solution leads to no more than $(1+\varepsilon)(3k_i + k_d)$ half-errors. This already implies near-MDS codes for deletion-only setting but still falls short for general insdel errors. We leave open the question whether an improved pseudo-distance definition can achieve an indexing solution with $(1+\varepsilon)k$ half-errors.

To achieve our main theorem, we developed a different strategy. Fortunately, it turned out that achieving a better indexing solution and the desired insdel codes does not require any changes to the definition of synchronization strings, the indexing approach itself, or the encoding scheme but solely required a very different decoding strategy. In particular, instead of guessing the position of symbols in a streaming manner, we consider more global, offline repositioning algorithms. We provide several such repositioning algorithms in Section 6. In particular, we give a simple global repositioning algorithm, for which the number of misdecodings goes to zero as the parameter ε of the utilized ε-synchronization string goes to zero, irrespective of how many insdel errors are applied.

Our global repositioning algorithms crucially build on another key-property that we prove holds for any ε-synchronization string S, namely, that there is no monotone matching between S and itself, which mismatches more than an ε-fraction of indices. Besides being used in our proofs, considering this ε-self-matching property has another advantage. We show that this property is achieved easier than the full ε-synchronization property and that indeed a random string satisfies it with good probability. This means that, in the context of error correcting codes, one can even use a simple uniformly random string as a "synchronization string." Last, we show that even an $n^{-O(1)}$-approximate $O(\frac{\log n}{\log(1/\varepsilon)})$-wise independent random string satisfies the desired ε-self-matching property that, using the celebrated small sample space constructions from Reference [32] also leads to a deterministic polynomial time construction for the index string.

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

1.4 Organization of the Article

The organization of this article 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 article in Section 2 together with a formal introduction of the two different error types as well as (efficient) error correcting codes and insdel codes. In Section 3, we formalize the indexing problem and provide solutions to it. 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 repositioning 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 1950s 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 that 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 $1 - O(\sqrt{\delta})$. However, for δ sufficiently close to one, one can achieve a constant rate of $O(\delta^2)$. Algebraic geometry codes suggested by Goppa in 1975 later led to error correcting codes that, for every $\varepsilon > 0$, achieve the optimal rate of $1-\delta-\varepsilon$ with an alphabet-size polynomial in ε while being able to efficiently correct from any δ fraction of half-errors [38].

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 [28], which do not require to fix a desired distance at the time of encoding, explicit expander codes [12, 36] that allow linear time encoding and decoding, polar codes [15, 17] that can approach Shannon's capacity polynomially fast, network codes [27] that allow intermediate nodes in a network to recombine codewords, and efficiently list decodable codes [14] that allow to list-decode codes of relative distance δ up to a fraction of about δ symbol substitutions.

While error correcting codes for insertions and deletions have also been intensely studied, our understanding of them is much less well developed. We refer to the 2002 survey by Sloan [35] on single-deletion codes, the 2009 survey by Mitzenmacher [30] on codes for random deletions, and the most general 2010 survey by Mercier et al. [29] 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 [26], and since then many bounds and constructions for such codes have been given. However, while essentially the same volume and sphere packing arguments as for regular codes show that there exist families of insdel codes capable of correcting a fraction δ of insdel errors with rate approaching $1 - \delta$, no efficient constructions anywhere close to this rate-distance tradeoff are known. Even the construction of efficient insdel codes over a constant alphabet with any (tiny) constant relative distance and any (tiny) constant rate had to wait until Schulman and Zuckerman gave the first such code in 1999 [34]. Over the couple of years preceding this work, Guruswami et al. provided new codes improving over this state-of-the-art in the asymptotically small or large noise regimes by giving the first codes that achieve a constant rate for noise rates going to one and codes that provide a rate going to one for an asymptotically small noise rate. In particular, Reference [16] gave the first efficient 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 [13]. In particular, these codes achieve a rate of $\Omega((1-\delta)^5)$ and $1-\tilde{O}(\sqrt{\delta})$, while being able to efficiently recover from a δ fraction of insertions and deletions in high-noise and high-rate regimes, respectively. These works put the 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 that we will use throughout the rest of the article.

2.1 String Notation and Edit Distance

String Notation. Let $S \in \Sigma^n$ 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, S^k is defined as k copies of S concatenated together and for integers $1 \leq i \leq j \leq n$, we denote the substring of S starting from the ith index through and including the jth one by $S[i,j]$. Such a substring is also called a *factor* of S. Further, for $i < 1$, we define $S[i,j] = \perp^{-i+1} \cdot S[1,j]$ where $\perp$ is a special symbol not included in Σ. We denote the substring from the ith index through, but not including, the jth index by $S[i,j)$. Substrings $S(i,j]$ and $S(i,j)$ are similarly defined. Finally, $S[i]$ denotes the ith symbol of string S and $|S|$ denotes the length of S. Occasionally, the alphabets we use are the 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 $a_i \in \Sigma_i$. We define the symbol-wise concatenation of two strings $S_1 \in \Sigma_1^n$ and $S_2 \in \Sigma_2^n$ as $S_1 \times S_2 \in (\Sigma_1 \times \Sigma_2)^n$ where $(S_1 \times S_2)[i] = (S_1[i], S_2[i])$ for all $1 \leq i \leq n$.

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* between two strings $c, c' \in \Sigma^*$ is the minimum number of insertions and deletions required to transform c into c' and is denoted by $\mathrm{ED}(c, c')$.

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, $\mathrm{ED}(c, c') = |c| + |c'| - 2 \cdot \mathrm{LCS}(c, c')$, where $\mathrm{LCS}(c, c')$ is the size of the longest common subsequence of c and c'.

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

Definition 2.2 (String Matching). Suppose that c and c' are two strings in Σ^* and $*$ is a symbol not included in Σ. Further, assume that there exist two strings τ_1 and τ_2 in $(\Sigma \cup \{*\})^*$ such that

$|\tau_1| = |\tau_2|$, $\text{del}(\tau_1) = c$, $\text{del}(\tau_2) = c'$, and $\tau_1[i] \approx \tau_2[i]$ for all $i \in \{1, \ldots, |\tau_1|\}$. Here, del is a function that deletes every $*$ in the input string and $a \approx b$ if $a = b$ or one of a or b is $*$. Then, we say that $\tau = (\tau_1, \tau_2)$ is a *string matching* between c and c' (denoted by $\tau : c \to c'$). We furthermore denote with $\text{sc}(\tau_i)$ the number of $*$'s in τ_i.

Note that the *edit distance* between strings $c, c' \in \Sigma^*$ is exactly equal to $\min_{\tau : c \to c'}\{\text{sc}(\tau_1) + \text{sc}(\tau_2)\}$.

2.2 Error Correcting Codes

Next, we give a quick summary of the standard definitions and formalism around error correcting codes. This is mainly for completeness, and we remark that readers already familiar with basic notions of error correcting codes might want to skip this part.

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

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

We remark that in addition to studying codes with decoding guarantees for worst-case error pattern as above, one can also look at more benign error models that assume a distribution over error patterns, such as errors occurring independently at random. In such a setting, one looks for codes that allow unique recovery for typical error patterns, i.e., one wants to recover the input with probability tending to 1 rapidly as the block length n grows. While synchronization strings might have applications for such codes as well, this article focuses exclusively on codes with good distance guarantees that tolerate an arbitrary (worst-case) error pattern.

Synchronization Errors. In addition to half-errors, we study *synchronization errors* that consist of *deletions*, that is, a symbol being removed without replacement, and *insertions*, where a new symbol from Σ is added anywhere. It is clear that **synchronization errors are strictly harsher and more general than half-errors** (see Section 1.1.1). The above formalism of codes, rate, and distance works equally well for synchronization errors if one replaces the Hamming distance with edit distance. Instead of measuring the number of symbol substitutions required to transform one string into another, *edit distance* measures the minimum number of insertions and deletions to do so. An insertion-deletion error correcting code, or *insdel code* for short, of relative distance δ is a set of codewords for which at least $2\delta n$ insertions and deletions are needed to transform any codeword into another. Such a code can correct any combination of less than δn insertions and deletions. We remark that it is possible for two codewords of length n to have edit distance up to $2n$.

Therefore, here we defined relative distance δ as the minimum edit distance of the code divided by $2n$. Formally, an insdel code C is an injective function that takes an input string $s \in (\Sigma')^{n'}$ over alphabet Σ' of length n' and generates a codeword $C(s) \in \Sigma^n$ of length n over alphabet Σ. The codeword length n is called the codes' block length. The rate of the code R is defined as $R = \frac{n' \log |\Sigma'|}{n \log |\Sigma|}$. The code distance $\Delta(C) = \min_{s,s'} \mathrm{ED}(C(s), C(s'))$ where ED indicates the edit distance function and the relative distance $\delta(C) = \frac{\Delta(C)}{2n}$.

Efficient Codes. In addition to codes with a good minimum distance, one furthermore wants efficient algorithms for the encoding and error-correction tasks associated with the code. Throughout this article, we say a code is efficient if it has encoding and decoding algorithms running in time polynomial in terms of the block length. While it is often not hard to show that random codes exhibit a good rate and distance, designing codes that can be decoded efficiently is much harder. We remark that most codes that can efficiently correct for symbol substitutions are also efficient for half-errors. For insdel codes the situation is slightly different. While it remains true that any code that can uniquely be decoded from any $\delta(C)$ fraction of deletions can also be decoded from the same fraction of insertions and deletions [26] doing so efficiently is often much easier for the deletion-only setting than the fully general insdel setting.

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

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

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

where $i = |\mathrm{del}(\tau_1[1, k])|$ and $j = |\mathrm{del}(\tau_2[1, k])|$.

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

The $\perp$ symbol here represents an "I don't know" response by the repositioning algorithm while a numerical output j for the ith symbol of S_τ should be interpreted as the algorithm guessing that $S_\tau[i]$ was at position j in string S prior to going through the insertion-deletion channel. We frequently refer to this procedure of guessing the position of the ith index symbol as *decoding index i*. One seeks algorithms that correctly decode as many indexes as possible. Naturally, one can only *correctly decode* index symbols that were *successfully transmitted.* We give formal definitions of both notions here.

Definition 3.2 (Correctly Decoded Index Symbol). An (n, δ)-indexing solution $(S, \mathcal{D}_S)$ decodes index j correctly under τ if $\mathcal{D}_S(S_\tau)$ outputs i for the jth received symbol and there exists a k such that $i = |\text{del}(\tau_1[1,k])|, j = |\text{del}(\tau_2[1,k])|, \tau_1[k] = S[i],$ 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 an index string S via $\tau = (\tau_1, \tau_2)$, we call the jth symbol $S_\tau[j]$ successfully transmitted if it stems from a symbol coming from S, i.e., if there exists a k such that $|\text{del}(\tau_2[1,k])| = j$ and $\tau_1[k] = \tau_2[k]$.

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

Definition 3.4 (Misdecodings of an (n, δ)-Indexing Solution). We say that an (n, δ)-indexing solution has at most k misdecodings if for any τ corresponding to at most $n\delta$ insertions and deletions, the number of successfully transmitted index symbols that are incorrectly decoded is at most k.

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

Definition 3.5 (Error-free Solution). We call $(S, \mathcal{D}_S)$ an error-free (n, δ)-indexing solution if for any error pattern τ, and for any element of S_τ, the repositioning algorithm $\mathcal{D}_S$ either outputs $\perp$ or correctly decodes that element. In other words, the repositioning algorithm never makes an incorrect guess, even for symbols that are inserted by the channel—it may just output $\perp$ for some of the successfully transmitted symbols.

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

Last, another very useful property of some (n, δ)-indexing solutions is that their decoding process operates in a streaming manner, i.e., the repositioning algorithm guesses the position of $S_\tau[j]$ independently of $S_\tau[j']$ where $j' > j$. While this property is not particularly useful for the error correcting block code application put forward in this article, it is an extremely important and strong property that 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 output of $\mathcal{D}_S$ for the ith element of the received string S_τ only depends on $S_\tau[1, i]$.

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

Overall, the important characteristics of an (n, δ)-indexing solution are (a) its alphabet size $|\Sigma|$, (b) the number of misdecodings it might bring about, (c) time complexity of the repositioning

Table 1. Properties and Quality of (n, δ)-indexing Solutions with S Being an ε-synchronization String

Algorithm	Type	Misdecodings	Error-free	Streaming	$\mathcal{D}_S(\cdot)$ Complexity
Section 5.2	ins/del	$(2+\varepsilon) \cdot n\delta$		✓	$O(n^4)$
Section 6.3	ins/del	$3\sqrt{\varepsilon} \cdot n$			$O\left(n^2/\sqrt{\varepsilon}\right)$
Section 6.4	del	$\varepsilon \cdot n\delta$		✓	$O(n)$
Section 6.5	ins	$(1+\varepsilon) \cdot n\delta$	✓		$O(n)$
Section 6.5	del	$\varepsilon \cdot n\delta$	✓		$O(n)$
Section 6.7	ins/del	$(1+\varepsilon) \cdot n\delta$		✓	$O(n^4)$

The alphabet size of string S is $\varepsilon^{-O(1)}$.

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

4 INSDEL CODES VIA INDEXING SOLUTIONS

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

To this end, we simply attach S symbol-by-symbol to every codeword of $\mathcal{C}$, i.e., obtain the code $\mathcal{C}_{out} = \{x \times S | x \in \mathcal{C}\}$. On the decoding end, we first guess the original positions of the symbols arrived using only the index portion of each received symbol and rearrange them accordingly. Positions where zero or multiple symbols are mapped to are considered as "?," i.e., ambiguous. We will refer to this procedure as *the rearrangement procedure.* Finally, the decoding algorithm $\mathcal{D}_\mathcal{C}$ for $\mathcal{C}$ is used over this rearranged string to finish decoding. These two straightforward algorithms are formally described as Algorithm 1 and Algorithm 2.

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

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

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

```
Input: $m$
1: $\tilde{m} = \mathcal{E}_C(m)$
Output: $\tilde{m} \times S$
```

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

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

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

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

Next, we formally state how a good (n, δ)-indexing solution $(S, \mathcal{D}_S)$ over alphabet Σ_S allows one to transform any regular ECC C with block length n over alphabet Σ_C that can efficiently correct half-errors, i.e., symbol substitutions 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 rearrangement procedure.

Theorem 4.2. *Given an (efficient) (n, δ)-indexing solution $(S, \mathcal{D}_S)$ over alphabet Σ_S with at most k misdecodings, and repositioning time complexity $T_{\mathcal{D}_S}$ and an (efficient) ECC 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 can obtain an insdel code by indexing codewords of C with S that can (efficiently) decode from up to $n\delta$ insertions and deletions. The rate of this code is at least*

$$\frac{R_C}{1 + \frac{\log|\Sigma_S|}{\log|\Sigma_C|}}.$$

The encoding complexity remains $T_{\mathcal{E}_C}$, the decoding complexity is $T_{\mathcal{D}_C} + T_{\mathcal{D}_S}$ 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 that can recover only from $n\delta + k$ erasures is sufficient to produce a code with same qualities.

Proof. Theorem 4.1 directly implies the distance quality. The encoding time complexity follows from the fact that concatenating the codeword with the index string takes linear time. The decoding time complexity is simply the sum of the running time of the algorithm $\mathcal{D}_S$ and the decoding complexity of code C. Finally, given that this transformation keeps the number of codewords as that of C and only increases the alphabet size, the rate of the resulting code is

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

□

Note that if one chooses Σ_C such that $\frac{\log|\Sigma_S|}{\log|\Sigma_C|} << 1$, the rate loss due to the attached symbols will be negligible. Using the observations outlined so far, one can obtain Theorem 1.1 as a consequence of Theorem 4.2.

4.1 Proof of Theorem 1.1

To prove this, we make use of the indexing solution that we will present later in Section 6.3. As outlined in Table 1, for any $\delta, \varepsilon' \in (0,1)$, Section 6.3 provides an indexing solution with $3n\sqrt{\varepsilon'}$ misdecodings and $O(n^2/\sqrt{\varepsilon'})$ repositioning time complexity. The alphabet size of the string in this solution is $\varepsilon'^{-O(1)}$. We also make use of the following near-MDS expander codes from Reference [12].

Theorem 4.3 (Guruswami and Indyk [12, Theorem 3]). *For every r, $0 < r < 1$, and all sufficiently small $\varepsilon > 0$, there exists an explicitly specified family of GF(2)-linear (also called additive) codes of rate r and relative distance at least $(1 - r - \varepsilon)$ over an alphabet of size $2^{O(\varepsilon^{-4}r^{-1}\log(1/\varepsilon))}$ such that codes from the family can be encoded in linear time and can also be (uniquely) decoded in linear time from a fraction e of errors and s of erasures provided $2e + s \leq (1 - r - \varepsilon)$.*

Given the δ and ε from the statement of Theorem 1.1, we choose $\varepsilon' = \frac{\varepsilon^2}{18^2}$ and consider the (n, δ)-indexing solution $(S, \mathcal{D}_S)$ as given in Section 6.3 (see line 2 of Table 1), which guarantees no more than $3n\sqrt{\varepsilon'} = \frac{n\varepsilon}{6}$ misdecodings. We then choose a near-MDS expander code C from Reference [12] (Theorem 4.3), which can efficiently correct up to $\delta_C = \delta + \frac{\varepsilon}{3}$ half-errors and has a rate of $R_C > 1 - \delta_C - \frac{\varepsilon}{3} = 1 - \delta - \frac{2\varepsilon}{3}$ over an alphabet Σ_C of size $\exp(\varepsilon^{-O(1)})$ that satisfies $\log|\Sigma_C| \geq \frac{3\log|\Sigma_S|}{\varepsilon}$. This ensures that the final rate is indeed at least $\frac{R_C}{1 + \frac{\log|\Sigma_S|}{\log|\Sigma_C|}} \geq R_C - \frac{\log|\Sigma_S|}{\log|\Sigma_C|} = 1 - \delta - 3\frac{\varepsilon}{3}$ and the fraction of insdel errors that can be efficiently corrected is $\delta_C - 2\frac{\varepsilon}{6} = \delta$. The encoding and decoding complexities follow Theorems 4.2 and 4.3 and the time complexity of the indexing solution as indicated in line 2 of Table 1. □

Theorem 1.1 is clearly optimal in its tradeoff between rate and efficiently decodable distance. As discussed in Section 1, its linear and quadratic encoding and decoding times are also optimal or hard to improve upon. (See the follow-up work [20] for an improved near-linear time decoding algorithm.) The only parameter that could be tightened is the dependence of the alphabet bit size on the parameter ε, which characterizes how close a code is to achieving an optimal rate/distance pair summing to one. Our transformation seems to inherently produce an alphabet bit size that is near-linear in $\frac{1}{\varepsilon}$ due to the rate loss stemming from alphabet expansion. For half-errors, ECCs based on algebraic geometry [38] achieving alphabet bit-size logarithmic in $\frac{1}{\varepsilon}$ are known, but their encoding and decoding complexities are higher. State of the art linear-time expander codes [33], which improve over [12], have an alphabet bit size, which is polylogarithmic in $\frac{1}{\varepsilon}$. Interestingly, a follow-up work by Haeupler et al. [23] shows that, as opposed to half-error ECCs, no such insdel code exist over alphabets that have sub-linear bit size in terms of $\frac{1}{\varepsilon}$.

5 SYNCHRONIZATION STRINGS

In this section, we formally define and develop ε-synchronization strings, which will be the base index string S in our (n, δ)-indexing solutions. As explained in Section 1.2, one can think of the prefixes $S[1, l]$ of an index string S as *codewords* encoding their length l, since the prefix $S[1, l]$, or a corrupted version of it, will be exactly all the position-related information that has been received by the time the lth symbol is communicated.

Definition 5.1 (Codewords Associated with an Index String). Given any index string S in a solution to an indexing problem, we define the set of codewords associated with S to be the set of prefixes of S, i.e., $\{S[1,l] \mid 1 \leq l \leq |S|\}$.

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

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

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

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

Next, we show that RSD is indeed a distance that 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 Reference [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 (see also Section 6.7).

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

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

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

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

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

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

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

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

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

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

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

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 string length n. We show the existence of ε-synchronization strings of arbitrary length for any $\varepsilon > 0$ using an alphabet size that is only polynomially large in $1/\varepsilon$. We remark that ε-synchronization strings can be seen as a strong generalization of square-free sequences in which any two neighboring substrings $S[i,j)$ and $S[j,k)$ only have to be different and not also far from each other in edit distance. Thue [37] famously showed the existence of arbitrarily large square-free strings over a ternary alphabet. Thue's methods for constructing such strings, however, turns out to be fundamentally too weak to prove the existence of ε-synchronization strings, for any constant $\varepsilon < 1$.

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

Lemma 5.6 (General Lovász Local Lemma). *Let $\{A_1, \ldots, A_n\}$ be a set of "bad" events. The directed graph $G(V,E)$ is called a dependency graph for this set of events if $V = \{1, \ldots, n\}$ and each event A_i is mutually independent of all the events $\{A_j : (i,j) \notin E\}$. Now, if there exists $x_1, \ldots, x_n \in [0,1)$ such that for all i, we have*

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

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

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

Theorem 5.7. *For any $\varepsilon \in (0,1)$ and $n \ge 1$, 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 the symbol-wise concatenation of two strings T and R, where T is simply the repetition of $0, \ldots, t-1$ for $t = \Theta(\frac{1}{\varepsilon^2})$, and R is a uniformly random string of length n over alphabet Σ. In other words, $S_i = (i \bmod t, R_i)$. We prove that S is an ε-synchronization string with positive probability by showing that there is a positive probability that S contains no *bad triple*, where (x,y,z) is a bad triple if $\mathrm{ED}(S[x,y), S[y,z)) \le (1-\varepsilon)(z-x)$.

First, note that a triple (x,y,z) for which $z - x \le t$ 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 $a_1a_2\ldots a_k$ be the longest common subsequence of $R[x,y)$ and $R[y,z)$. It is straightforward to see that $\mathrm{ED}(R[x,y), R[y,z)) = (y-x) + (z-y) - 2k = z - x - 2k$. Since (x,y,z) is a bad triple, we have that $z - x - 2k \le (1-\varepsilon)(z-x)$, which means that $k \ge \frac{\varepsilon}{2}(z-x)$. With this observation in mind, we say that $R[x, z)$ is a *bad interval* if it contains a subsequence $a_1 a_2 \dots a_k a_1 a_2 \dots a_k$ such that $k \geq \frac{\varepsilon}{2}(z - x)$.

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

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

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

To show that there is a non-zero probability that the uniformly random string R contains no bad interval I of size t or larger, we use the general Lovász local lemma stated in Lemma 5.6. Note that the badness of interval I is mutually independent of the badness of all intervals that do not intersect I. We need to find real numbers $x_{p,q} \in [0, 1)$ corresponding to intervals $R[p, q)$ for which

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

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

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

or more clearly, for all $l' \in \{1, 2, \dots, n\}$ satisfies

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

which means that

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

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 $l' \in \{1, 2, \dots, n\}$, it suffices to find a D such that

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

To this end, let

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

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

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

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

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

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

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

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

$$= \frac{D}{1-\frac{1}{\varepsilon}\sum_{l=t}^{\infty}(l+1)\cdot(D^{-\varepsilon})^l} \tag{4}$$

$$< \frac{D}{1-\frac{1}{\varepsilon}\frac{2t(D^{-\varepsilon})^t}{(1-D^{-\varepsilon})^2}} \tag{5}$$

$$= \frac{D}{1-\frac{2}{\varepsilon^3}\frac{D^{-\frac{1}{\varepsilon}}}{(1-D^{-\varepsilon})^2}}. \tag{6}$$

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 (5) is a result of the following equality for $x < 1$:

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

One can see that for $D = 7$, $\max_\varepsilon\{\frac{2}{\varepsilon^3}\frac{D^{-\frac{1}{\varepsilon}}}{(1-D^{-\varepsilon})^2}\} < 0.9$, and therefore step (3) is legal and step (6) can be upper-bounded by a constant. Hence, $L = \Theta(1)$ and the proof is complete. □

Remarks on the alphabet size: Theorem 5.7 shows that for any $\varepsilon > 0$ there exists an ε-synchronization string over alphabets of size $O(\varepsilon^{-4})$. A polynomial dependence on ε is also necessary. In particular, there do not exist any ε-synchronization string over alphabets of size smaller than ε^{-1}. In fact, any $\frac{1}{\varepsilon}$-long substring of an ε-synchronization string has to contain completely distinct elements. This can be easily proven as follows: For the sake of contradiction let $S[i, i+\varepsilon^{-1})$ be a substring of an ε-synchronization string where $S[j] = S[j']$ for $i \leq j < j' < i + \varepsilon^{-1}$. Then, $\mathrm{ED}(S[j], S[j+1, j'+1))) = j' - j - 1 = (j'+1-j) - 2 \leq (j'+1-j)(1-2\varepsilon)$. 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 $O(\varepsilon^{-2})$ (see Reference [4]). It seems much harder to improve the alphabet size to $o(\varepsilon^{-2})$, and we are not convinced that it is possible. This work thus leaves open the interesting question of closing the quadratic gap between $O(\varepsilon^{-2})$ and $\Omega(\varepsilon^{-1})$ from either side.

Theorem 5.7 also implies an efficient randomized construction.

Lemma 5.8. *There exists a randomized algorithm that for any $\varepsilon > 0$ and n constructs an ε-synchronization string of length n over an alphabet of size $O(\varepsilon^{-4})$ with expected running time $O(n^7)$.*

Proof. Using the algorithmic framework for the Lovász local lemma given by Moser and Tardos [31] (or the extensions by Haeupler et al. [21]), one can get such a randomized algorithm from the proof in Theorem 5.7. The algorithmic LLL method of Reference [31] works as follows: It first generates a random string over an alphabet Σ (of size $C\varepsilon^{-4}$ for a sufficiently large C). It then checks all $O(n^2)$ substrings for a bad event (i.e., a substring that violates the LLL condition). If a violating interval is found, then all symbols within that interval are resampled. This process repeats until no violations are remained.

To find a violation, one can compute the edit distance between all neighboring substrings. This edit distance computation can be done in $O(n^2)$ time using the classic Wagner-Fischer dynamic programming algorithm. Therefore, the total running time would be the number of times that resampling is needed times $O(n^5)$. We bound the number of resampling rounds using the following theorem from Reference [31].

Theorem 5.9 (Theorem 1.2 of Reference [31]). *Let $\mathcal{P}$ be a finite set of mutually independent random variables in a probability space. Let $\mathcal{A}$ be a finite set of events determined by these variables. If there exists an assignment of reals $x : \mathcal{A} \to (0, 1)$ such that*

$$\forall A \in \mathcal{A} : \Pr[A] \leq x(A) \prod_{B \in \Gamma_{\mathcal{A}}(A)} (1 - x(B)),$$

then there exists an assignment of values to the variables P not violating any of the events in A. Moreover the randomized algorithm described above resamples an event $A \in \mathcal{A}$ at most an expected $x(A)/(1-x(A))$ times before it finds such an evaluation. Thus, the expected total number of resampling steps is at most $\sum_{A \in \mathcal{A}} \frac{x(A)}{1-x(A)}$.

In the proof of Theorem 5.7, we showed that the real value $x(A) = D^{-\varepsilon l}$ for an interval of length l would satisfy the condition of Theorem 5.9 for an adequately large constant D like $D = 7$. Thus, the expected number of total resamplings is

$$\sum_{i<j} \frac{x(A_{i,j})}{1 - x(A_{i,j})} = \sum_{i<j} \frac{7^{-\varepsilon(j-i)}}{1 - 7^{-\varepsilon(j-i)}} = \sum_{i<j} \frac{1}{7^{\varepsilon(j-i)} - 1} = O_\varepsilon(n^2),$$

where $A_{i,j}$ represents the event that the substring indicated by $[i, j]$ is bad. This gives an expected running time of $O(n^7)$ overall, as claimed. □

Last, since synchronization strings can be repositioned in a streaming fashion, they can be used in many important applications where the length of the required synchronization string is not known in advance (see References [22, 24]). In such a setting it is advantageous to have an infinite synchronization string over a fixed alphabet. In particular, since every substring of an ε-synchronization string is also an ε-synchronization string by definition, having an infinite ε-synchronization string also implies the existence for every length n, i.e., Theorem 5.7. Interestingly, a simple argument shows that the converse is true as well, i.e., the existence of an ε-synchronization string for every length n implies the existence of an infinite ε-synchronization string over the same alphabet.

Lemma 5.10. *For any $\varepsilon \in (0,1)$ there exists an infinite ε-synchronization string over an alphabet of size $\Theta(1/\varepsilon^4)$.*

Proof. Fix any $\varepsilon \in (0,1)$. According to Theorem 5.7 there exist an alphabet Σ of size $O(1/\varepsilon^4)$ such that there exists at least one ε-synchronization strings over Σ for every length $n \in \mathbb{N}$. We will define an infinite synchronization string $S = s_1 \cdot s_2 \cdot s_3 \ldots$ with $s_i \in \Sigma$ for any $i \in \mathbb{N}$ for which the ε-synchronization property holds for any neighboring substrings. We define this string inductively. In particular, we fix an ordering on Σ and define $s_1 \in \Sigma$ to be the first symbol in this ordering such that an infinite number of ε-synchronization strings over Σ starts with s_1. Given that there is an infinite number of ε-synchronization over Σ such an s_1 exists. Furthermore, the subset of ε-synchronization strings over Σ that start with s_1 is infinite by definition, allowing us to define $s_2 \in \Sigma$ to be the lexicographically first symbol in Σ such there exists an infinite number of ε-synchronization strings over Σ starting with $s_1 \cdot s_2$. In the same manner, we inductively define s_i to be the lexicographically first symbol in Σ for which there exists an infinite number of ε-synchronization strings over Σ starting with $s_1 \cdot s_2 \cdot \ldots \cdot s_i$. To see that the infinite string defined in this manner does indeed satisfy the edit distance requirement of the ε-synchronization property defined in Definition 5.4, we note that for every $i < j < k$ with $i, j, k \in \mathbb{N}$ there exists, by definition, an ε-synchronization string and, in fact, an infinite number of them, which contains $S[1,k]$ and thus also $S[i,k]$ as a substring implying that indeed $ED\left(S[i,j), S[j,k)\right) > (1-\varepsilon)(k-i)$ as required. Our definition thus produces the unique lexicographically first infinite ε-synchronization string over Σ. □

We remark that any string produced by the randomized construction of Lemma 5.8 is guaranteed to be a correct ε-synchronization string and the only randomized aspect of the algorithm is its running time. This randomized synchronization string construction is furthermore only needed once as a pre-processing step. The encoder or decoder of any resulting error correcting codes do not require any randomization. Furthermore, in Section 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.

It nonetheless remains interesting to obtain fast deterministic constructions of finite and infinite ε-synchronization strings. In a subsequent work [22], we achieve such efficient deterministic constructions for ε-synchronization strings. Our constructions in Reference [22] even produce the infinite ε-synchronization string S proven to exist by Lemma 5.10, which is much less explicit: While for any n and ε an ε-synchronization string of length n can in principle be found using an exponential time enumeration, there is no straightforward algorithm that follows the proof of Lemma 5.10 and given an $i \in \mathbb{N}$ produces the ith symbol of such an S in a finite amount of time (bounded by some function in i). Our constructions in Reference [22] require significantly more work but in the end lead to an explicit deterministic construction of an infinite ε-synchronization string for any $\varepsilon > 0$ for which the ith symbol can be computed in only $O(\log i)$ time—thus satisfying one of the strongest notions of constructiveness that can be achieved.

5.2 Repositioning Algorithm for ε-Synchronization Strings

We now provide an algorithm for repositioning synchronization strings, i.e., an algorithm that can form a solution to the indexing problem along with ε-synchronization strings. In the beginning of Section 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 repositioning scheme for guessing the position of a received symbol would be finding the prefix of the index string with the closest relative suffix distance to the string received thus far. We call this algorithm *the minimum relative suffix distance decoding algorithm.*

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

Theorem 5.11. *Any ε-synchronization string of length n along with the minimum relative suffix distance decoding algorithm form a solution to (n, δ)-indexing problem that guarantees $\frac{2}{1-\varepsilon}n\delta$ or less misdecodings. This repositioning algorithm is streaming and can be implemented in a way that works in $O(n^4)$ time.*

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

Definition 5.12 (Error Count Function). Let S be an index string sent over an insertion-deletion channel. We denote the *error count from index i to index j* with $\mathcal{E}(i, j)$ and define it to be the number of insdels applied to S from the moment $S[i]$ is sent until the moment $S[j]$ is sent. $\mathcal{E}(i, j)$ would count the deletion of $S[j]$, however, it would *not* count the deletion of $S[i]$.

Definition 5.13 (Relative Suffix Error Density). Let string S be sent over an insertion-deletion channel and let $\mathcal{E}$ denote the corresponding error count function. We define the *relative suffix error density* of the communication as

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

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

Lemma 5.14. *Let string S be sent over an insertion-deletion channel and the corrupted version of it S' be received on the other end. The relative suffix distance between S and S',* $\mathrm{RSD}(S, S')$*, is at most the relative suffix error density of the communication.*

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

$$\begin{aligned}
\mathrm{RSD}(S, S') &= \max_{k>0} \frac{\mathrm{ED}(S(|S| - k, |S|], S'(|S'| - k, |S'|])}{2k} && (7)\\
&= \max_{k>0} \frac{\min_{\tau: S(|S|-k, |S|] \to S'(|S'|-k, |S'|]}\{sc(\tau_1) + sc(\tau_2)\}}{2k} && (8)\\
&\leq \max_{k>0} \frac{2(sc(\tilde{\tau}_1^k) + sc(\tilde{\tau}_2^k))}{2k} \leq \text{Relative Suffix Error Density}, && (9)
\end{aligned}$$

where $\tilde{\tau}^k$ is $\tilde{\tau}$ limited to its suffix corresponding to $S(|S| - k, |S|]$. More precisely, the suffix of $\tilde{\tau}$ that starts from the position following the one that corresponds to $S[|S| - k]$. (We remind the reader that the notions of string matching and $sc(\cdot)$ are defined in Definition 2.2.) Note that steps (7) and (8) follow from the definitions of edit distance and relative suffix distance. Moreover, to verify step (9), one has to note that one single insertion or deletion on the k-element suffix of a string may result in a string with k-element suffix of edit distance two of the original string's k-element suffix; one stemming from the inserted/deleted symbol and the other one stemming from a symbol appearing/disappearing at the beginning of the suffix to keep the size of suffix k. □

A key consequence of Lemma 5.14 is that if an ε-synchronization string is being sent over an insertion-deletion channel and at some step the relative suffix error density corresponding to errors is smaller than $\frac{1-\varepsilon}{2}$, the relative suffix distance of the sent string and the received one at that point is smaller than $\frac{1-\varepsilon}{2}$; therefore, as RSD of all pairs of codewords associated with an ε-synchronization string are greater than $1-\varepsilon$, the receiver can correctly decode the index of—or guess the position of—the last received symbol of the communication by simply finding the codeword (i.e., prefix of the synchronization string) with minimum relative suffix distance to the string received so far.

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

Lemma 5.15. *Let ε-synchronization string S be sent over an insertion-deletion 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 $t \geq 1$, 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 $\{0, 1, \ldots, n\}$ as follows:

$$\Phi(i) = \max\left\{0, \max_{1\leq s\leq i}\left\{\frac{t\cdot\mathcal{E}(i-s,i)}{1-\varepsilon} - s\right\}\right\}.$$

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 changes as follows: $\Phi(i) = \max\{0, \Phi(i-1) - 1\}$.
(2) If $\mathcal{E}(i-1,i) = k$, i.e., the adversary inserts or deletes k symbols in the interval starting right after the moment $S[i-1]$ is sent and ending at when $S[i]$ is sent, then the value of Φ increases by $\frac{tk}{1-\varepsilon} - 1$, i.e., $\Phi(i) = \Phi(i-1) + \frac{tk}{1-\varepsilon} - 1$.
(3) If $\Phi(i) = 0$, then the relative suffix error density of the string that is received when $S[i]$ arrives at the receiving side is not larger than $\frac{1-\varepsilon}{t}$.

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

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

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

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

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

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

(3) And finally,

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

These finish the proof of the lemma. □

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

Proof of Theorem 5.11. As the adversary is allowed to insert or delete up $n\delta$ symbols, by Lemma 5.15, 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 5.14, there are at most $\frac{2n\delta}{1-\varepsilon}$ misdecoded successfully transmitted symbols.

Further, we remark that this algorithm can be implemented in $O(n^4)$ time. Using dynamic programming, we keep track of the edit distance of any substring of S, like $S[i,j]$ to any substring of S', like $S'[i',j']$ as symbols arrive. Of course, at any point in time, this can only be computed for all i,j,i',j', where j' is no larger than the current length of the communication. Given that each distance can be computed using a constant number of sub-problems, this would take a total of $O(n^4)$ time.

Then, to decode each index symbol like $S'[l']$ (i.e., guess its original position), we can find the codeword with minimum relative suffix distance to $S'[1,l']$ by calculating the relative suffix distance of $S'[1,l']$ to all n codewords using the edit distance data described above. Finding the suffix distance of $S'[1,l']$ and a codeword like $S[1,l]$ is simply done by minimizing $\frac{ED(S(l-k,l],S'(l'-k,l'])}{k}$ over all k. This can be done in $O(n)$ time. With an $O(n^4)$ total time needed to keep track of of the edit distance between substrings and an $O(n^3)$ suffix distance computation as mentioned above, we have shown that the decoding process can be implemented in a total of $O(n^4)$ time. Finally, this algorithm clearly satisfies streaming property as it decodes indices of arrived symbols merely using the symbols that have arrived earlier. □

We remark that by taking $\varepsilon = o(1)$, one can obtain a solution to the (n,δ)-indexing problem with a misdecoding guarantee of $2n\delta(1+o(1))$, which, using Theorem 4.1, results into a translation of $n\delta$ insertions and deletions into $n\delta(5+o(1))$ half-errors.

In Section 6.7 , we show that this guarantee of the min-distance-decoder can be slightly improved to $n\delta(3+o(1))$ half-errors, at the cost of some simplicity. In particular, one can go beyond an RSD distance of $\frac{1-\varepsilon}{2}$ by considering the RSPD, which was introduced in Reference [2], as an alternative distance measure. RSPD can act as a stand-in metric for the minimum-distance decoder and lead to the above-mentioned slightly improved decoding guarantees, despite neither being symmetric nor satisfying the triangle inequality. More precisely, for any set of $k = k_i + k_d$ insdel errors consisting of k_i insertions and k_d deletions, the RSPD-based indexing solution leads to at most $(1+\varepsilon)(3k_i+k_d)$ half-errors, which does imply "near-MDS" codes for deletion-only channels but still falls short for general insdel errors.

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

6 MORE ADVANCED REPOSITIONING ALGORITHMSAND ε-SELF-MATCHING PROPERTY

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 a guarantee of $\frac{2n\delta}{1-\varepsilon}$ misdecodings, which, by taking ε adequately small, tends to $2n\delta$. As explained in Section 1.3, such a guarantee falls short of giving Theorem 1.1. In this section, we thus provide a variety of more advanced repositioning algorithms that provide better decoding guarantees, in particular, achieve a misdecoding fraction that goes to zero as ε tends 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 non-trivial pairs (i.e., those that do not correspond to the same element of the string) is limited. 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 repositioning algorithm for insdel errors that basically works by finding monotone matchings between the received string and the synchronization string. Using the ε-self-matching property, we show that this algorithm guarantees $O(n\sqrt{\varepsilon})$ misdecodings. This algorithm works in $O(n^2/\sqrt{\varepsilon})$ time and is exactly what we need to prove our main theorem.

In Sections 6.4 and 6.5, we provide two simpler linear-time repositioning algorithms that solve the indexing problem under the assumptions that the adversary can only delete symbols or only insert symbols. These algorithms not only guarantee asymptotically optimal $\frac{\varepsilon}{1-\varepsilon}n\delta$ misdecodings but are also error-free. In Section 6.6, we present linear-time near-MDS insertion-only and deletion-only codes that can be derived by these repositioning algorithms.

Finally, in Section 6.7, we present an improved version of the minimum RSD decoding algorithm that achieves a better misdecoding guarantee by replacing RSD with a similar pseudo-distance.

See Table 1 for a break down of the repositioning schemes presented in this article, the type of error under which they work, the number of misdecodings they guarantee, whether they are error-free or streaming, and their repositioning time complexities.

6.1 ε-Self-matching Property

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

Definition 6.1 (Monotone Matchings). A monotone matching between S and S' is a set of pairs of indexes like

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

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

We now point out a key property of synchronization strings that will be broadly used in our repositioning algorithms in Theorem 6.2, which, basically, states that two large similar subsequences of an ε-synchronization string cannot disagree on many positions. More formally, let $M = \{(a_1, b_1), \ldots, (a_m, b_m)\}$ be a monotone matching between S and itself. We call the pair (a_i, b_i) a *good pair* or a *good match* if $a_i = b_i$ and a *bad pair* or a *bad match* otherwise. Then:

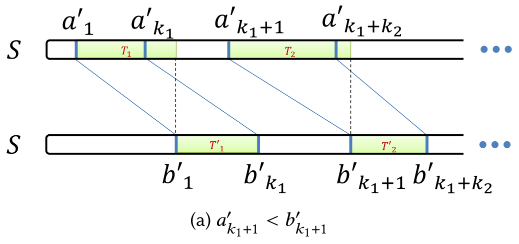
Theorem 6.2. *Let S be an ε-synchronization string of size n and $M = \{(a_1, b_1), \ldots, (a_m, b_m)\}$ be a monotone matching of size m from S to itself containing g good pairs and b bad pairs. Then,*

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

Proof. Let $(a'_1, b'_1), \ldots, (a'_b, b'_b)$ indicate the set of bad pairs in M indexed as $a'_1 < \cdots < a'_b$ and $b'_1 < \cdots < b'_b$. Without loss of generality, assume that $a'_1 < b'_1$. Let k_1 be the largest integer such that $a'_{k_1} < b'_1$. Then, the pairs $(a'_1, b'_1), \ldots, (a'_{k_1}, b'_{k_1})$ form a common subsequence of size k_1 between $T_1 = S[a'_1, b'_1)$ and $T'_1 = S[b'_1, b'_{k_1}]$. Now, the synchronization string guarantee implies that

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

Note that the monotonicity of the matching guarantees that there are no good matches occurring on indices covered by T_1 and T'_1, i.e., $a'_1, \ldots, b'_{k_1}$. One can repeat the very same argument for



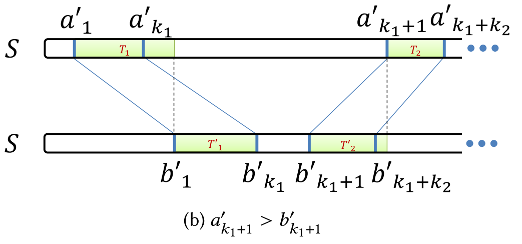


Fig. 1. Pictorial representation of T_2 and T'_2.

the remaining bad matches to rule out bad matches $(a'_{k_1+1}, b'_{k_1+1}), \ldots, (a'_{k_1+k_2}, b'_{k_1+k_2})$ for some k_2 having the following inequality guaranteed:

$$k_2 < \frac{\varepsilon(|T_2| + |T'_2|)}{2}, \tag{10}$$

where

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

For a pictorial representation see Figure 1.

Continuing the same procedure, one can find $k_1, \ldots, k_l$, $T_1, \ldots, T_l$, and $T'_1, \ldots, T'_l$ for some l. Summing up all inequalities of form of Equation (10), we will have

$$\sum_{i=1}^{l} k_i < \frac{\varepsilon}{2} \cdot \left(\sum_{i=1}^{l} |T_i| + \sum_{i=1}^{l} |T'_i| \right). \tag{11}$$

Note that $\sum_{i=1}^{l} k_i = b$ and T_is are mutually exclusive and do not contain any good pairs. The same holds for T'_is. Hence, $\sum_{i=1}^{l} |T_i| \le n-g$ and $\sum_{i=1}^{l} |T'_i| \le n-g$. All these along with Equation (11) give that

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

□

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 in the following way:*

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

Proof of Theorem 6.4 (a). This part is a straightforward consequence of Theorem 6.2. □

Proof of Theorem 6.4 (b). Assume by contradiction that there are $i < j < k$ such that $ED(S[i,j), S[j,k)) \le (1-\varepsilon)(k-i)$. Then, $LCS(S[i,j), S[j,k)) \ge \frac{k-i-(1-\varepsilon)(k-i)}{2} = \frac{\varepsilon}{2}(k-i)$. The corresponding pairs of such longest common subsequence form a monotone matching of size $\frac{\varepsilon}{2}(k-i)$, which contradicts $\frac{\varepsilon}{2}$-self-matching property of S. □

The repositioning algorithms we will propose for ε-synchronization strings in Sections 6.3, 6.4, and 6.5 only make use of the ε-self-matching property of ε-synchronization strings. We now define ε-bad elements.

Definition 6.5 (ε-bad Element). We call element k of string S an ε-bad element if there exists a factor $S[i,j]$ of S with $i \leq k \leq j$ where $S[i,j]$ does not satisfy the ε-self-matching property. In this case, we also say that element k blames interval $[i,j]$.

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

Lemma 6.6. *Let S be an ε-self-matching string of length n. Then, for any $3\varepsilon < \varepsilon' < 1$, at most $\frac{3n\varepsilon}{\varepsilon'}$ many elements of S can be ε'-bad.*

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

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

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

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

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

As the final remark on the ε-self-matching property and its relation with the more strict ε-synchronization property, we show that using the minimum RSD decoder together with an ε-self-matching string leads to indexing solutions with a guarantee on the misdecoding count, which is only slightly weaker than the guarantee obtained by ε-synchronization strings. To do so, we first show that the $(1-\varepsilon)$ RSD distance property of prefixes holds for any non-ε-bad element in any arbitrary string in Lemma 6.7. Then, using Lemma 6.7 and Lemma 6.6, we bound above the number of misdecodings that may occur using a minimum RSD decoder along with an ε-self-matching string in Theorem 6.8.

Lemma 6.7. *Let S be an arbitrary string of length n and $1 \leq i \leq n$ be such that ith element of S is not ε-bad. Then, for any $j \neq i$, $\mathrm{RSD}(S[1,i], S[1,j]) > 1 - \varepsilon$.*

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

We remind the reader from Section 2.1 that, in the case where $2j - i + 1 < 1$, $S[2j - i + 1, j]$ is defined as $\perp^{i-2j} \cdot S[1, j]$. In this case, since i is not ε-bad, $ED(S[1, j], S[j + 1, i]) > (1 - \varepsilon) \cdot i$. Since $\perp$ is a special symbol that does not belong to the alphabet, $ED(S[2j - i + 1, j], S[j + 1, i]) = ED(S[1, j], S[j + 1, i]) + (i - 2j) > (1 - \varepsilon)i + (i - 2j) > (1 - \varepsilon) \cdot 2(i - j)$, which implies that $RSD(S[1, i], S[1, j]) > 1 - \varepsilon$. □

Theorem 6.8. *Using any ε-self-matching string along with minimum RSD algorithm, one can solve the (n, δ)-indexing problem with no more than $n(4\delta + 6\varepsilon)$ misdecodings.*

Proof. Note that applying Lemma 6.6 for ε' gives that there are at most $\frac{3n\varepsilon}{\varepsilon'}$ indices in S that are ε'-bad. Lemma 6.7 implies that for all indices i that are not ε'-bad, the desired RSD guarantee for minimum-RSD decoding holds, i.e., $RSD(S[1, i], S[1, j]) > 1 - \varepsilon'$ for all $i \neq j$. Therefore, such symbols are misdecoded only if the relative suffix error density exceeds $\frac{1-\varepsilon'}{2}$ upon their arrival—which, as shown in Lemma 5.15, can happen for no more than $\frac{2n\delta}{1-\varepsilon'}$ of positions. Therefore, this solution for the (n, δ)-indexing problem can contain at most $n(\frac{3\varepsilon}{\varepsilon'} + \frac{2\delta}{1-\varepsilon'})$ 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 Efficient Polynomial Time Construction of ε-Self-matching Strings

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

Theorem 6.9. *For any $\alpha > 0$ and any sufficiently small $\varepsilon > 0$, a random string over an alphabet of size $\varepsilon^{-(2+\alpha)}$ satisfies the ε-self-matching property with high probability.*

Proof. Let S be a random string on alphabet Σ of size $|\Sigma| = \varepsilon^{-(2+\alpha)}$. For S to not satisfy the ε-self-matching property, there has to be a self-matching between S and itself with $n\varepsilon$ bad pairs. Note that there are no more than $\binom{n}{n\varepsilon}^2$ possible such self-matchings. Further, for any such potential self-matching, the probability of it actually constituting a self-matching is $\frac{1}{|\Sigma|^{n\varepsilon}}$. Therefore, using the union bound, the probability of S not satisfying the ε-self-matching property is bounded above by

$$\binom{n}{n\varepsilon}^2 \cdot \frac{1}{|\Sigma|^{n\varepsilon}} \leq \left(\frac{ne}{n\varepsilon}\right)^{2n\varepsilon} \cdot \frac{1}{|\Sigma|^{n\varepsilon}} \leq \left(\frac{e}{\varepsilon\sqrt{|\Sigma|}}\right)^{2n\varepsilon} = \left(e\varepsilon^{\frac{\alpha}{2}}\right)^{2n\varepsilon} = \left(e^2\varepsilon^{\alpha}\right)^{n\varepsilon}.$$

Note that for a sufficiently small ε, $e^2\varepsilon^{\alpha} < 1$ and thus, $(e^2\varepsilon^{\alpha})^{\varepsilon} < 1$. Therefore, the probability of S not satisfying the ε-self-matching property approaches zero exponentially as n grows. □

As the next step, we prove a similar claim for strings of length n whose symbols are chosen from an $n^{-O(1)}$-approximate $\Theta(\frac{\log n}{\log(1/\varepsilon)})$-wise independent [32] 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 [32]. The proof of Theorem 6.10 follows a similar strategy as was used in Reference [3] to derandomize the constructive Lovász local lemma. In particular, the crucial idea, that is stated in Lemma 6.11, 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 obstructions, no large obstructions would exist.

Theorem 6.10. *For sufficiently small ε amd sufficiently large constants c and c_0, a n^{-c_0}-approximate $\frac{c \log n}{\log(1/\varepsilon)}$-wise independent random string of size n on an alphabet of size $O(\varepsilon^{-2.01})$ satisfies the ε-matching property with high probability.*

Proof. Let S be a pseudo-random string of length n with n^{-c_0}-approximate $\frac{c \log n}{\log(1/\varepsilon)}$-wise independent symbols. We use the following to show that S would violate the ε-self-matching property only if there is a dense matching between two small substrings of S.

Lemma 6.11. *Consider string I of length $l > 100m$ and a self-matching over it of size $n\varepsilon$ consisting only of bad pairs. There exist (not necessarily disjoint) substrings I_1 and I_2 of I, both of size m where $0.99\frac{m\varepsilon}{2}$ of the pairs of the matching lie between I_1 and I_2—i.e., have one endpoint in I_1 and one endpoint in I_2.*

If S does not satisfy the ε-self-matching property, then there exists a matching with $n\varepsilon$ bad pair between S and itself. By applying Lemma 6.11 over this matching with $m = \frac{c \log n}{\varepsilon \log(1/\varepsilon)}$ for some c that we fix later, we have that there are two (not necessarily distinct) substrings of S of size $\frac{c \log n}{\varepsilon \log(1/\varepsilon)}$ where $\frac{0.99c \log n}{2 \log(1/\varepsilon)}$ pairs of S's self-matching lie between them. Therefore, to bound above the probability of S not satisfying the ε-self-matching property, we simply bound above the probability of the existence of a self-matching over S with $\frac{0.99c \log n}{2 \log(1/\varepsilon)}$ bad pairs where first endpoints of all pairs lie in a substring of length $\frac{c \log n}{2 \log(1/\varepsilon)}$ and their second endpoints lie in another (not necessarily disjoint) substring of length $\frac{c \log n}{2 \log(1/\varepsilon)}$.

To do so, similar to Theorem 6.9, we take advantage of the union bound. Note that there are no more than n^2 choices of pairs of intervals of length $\frac{c \log n}{\varepsilon \log(1/\varepsilon)}$ in S. Therefore, there can only be up to

$$\begin{aligned} n^2 \binom{\varepsilon^{-1} c \log n / \log(1/\varepsilon)}{0.99c \log n / 2 \log(1/\varepsilon)}^2 &\le n^2 \cdot \left(\frac{2e}{0.99\varepsilon}\right)^{2 \times \frac{0.99c \log n}{2 \log(1/\varepsilon)}} \le n^2 \cdot \left(2.04 e \varepsilon^{-1}\right)^{\frac{0.99c \log n}{\log(1/\varepsilon)}} \\ = \quad n^2 \cdot (2.04e)^{\frac{0.99c \log n}{\log(1/\varepsilon)}} n^{0.99c} &= n^{2 + 0.99c + \frac{0.99c \log(2.04e)}{\log(1/\varepsilon)}} \end{aligned}$$

potential sets of bad pairs that meet the above mentioned criteria.

We now bound above the probability of the realization of a bad matching over a specific set of endpoints as described above. More precisely, we have two sets of $k = \frac{0.99c \log n}{2 \log(1/\varepsilon)}$ locations in S denoted by $p_1 < p_2 < \cdots < p_k$ and $q_1 < q_2 < \cdots < q_k$. If the randomness is drawn out of a $2k$-wise independent distribution, then the probability of a bad matching being realized at this locations is $\frac{1}{|\Sigma|^k}$. Note that our distribution here is n^{-c_0}-approximate $\frac{c \log n}{\log(1/\varepsilon)}$-wise independent and $2k < \frac{c \log n}{\log(1/\varepsilon)}$. Therefore, the probability of a bad matching forming at $\{(p_i, q_i)\}_{i=1}^n$ is no more than

$$\frac{1}{|\Sigma|^k} + n^{-c_0} = \frac{1}{\varepsilon^{-2.01 \cdot \frac{0.99c \log n}{2 \log(1/\varepsilon)}}} + n^{-c_0} = n^{-2.01 \cdot \frac{0.99c}{2}} + n^{-c_0} \le 2n^{-2.01 \cdot \frac{0.99c}{2}}.$$

For the last step to hold, we set c_0 in a way that $c_0 > c$.

All in all, the probability of S not satisfying the ε-self-matching probability is at most

$$n^{2 + 0.99c + \frac{0.99c \log(2.04e)}{\log(1/\varepsilon)}} \cdot 2n^{-2.01 \cdot \frac{0.99c}{2}} = 2n^{2 + 0.99c\left(\frac{\log(2.04e)}{\log(1/\varepsilon)} - 0.005\right)}.$$

For a sufficiently small ε, $\frac{\log(2.04e)}{\log(1/\varepsilon)} - 0.005 < 0$. Therefore, if c is a sufficiently large constant, then the exponent of n in the above expression is negative. Therefore, the probability of S not satisfying the ε-self-matching probability is polynomially small in terms of n and vanishes as n grows. □

Proof of Lemma 6.11. Let M be a monotone matching of size $l\varepsilon$ or more between S and itself containing only bad edges. We chop S into $\frac{l}{m}$ intervals of size m. On the one hand, the size of M is greater than $l\varepsilon$ and, on the other hand, we know that the size of M is exactly $\sum_{i,j} |E_{i,j}|$, where

$E_{i,j}$ denotes the number of edges between interval i and j. Thus,

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

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

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

To analyze more accurately, if l is not divisible by m, then 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 Reference [32] Theorem 6.10 directly leads to a deterministic algorithm for finding a string of size n that satisfies the ε-self-matching property.

Theorem 6.12. *There is a deterministic algorithm running in $n^{O(1)}$ time that finds a string of length n satisfying ε-self-matching property over an alphabet of size $O(\varepsilon^{-2.01})$.*

Proof. To do so, one simply has to do a brute force search over all possible points in the sample space of some $n^{-O(1)}$-approximate $\frac{c \log n}{\log(1/\varepsilon)}$-wise independent string of length n and find one that satisfies the ε-self-matching property. Reference [32] gives constructions of n random variables that are δ-approximate k-wise independent with sample spaces of size $\exp(k + \log \frac{1}{\delta} + \log\log n)$. (See Lemma 4.2 of Reference [32] for more information.) Therefore, doing so would take

$$\exp\left(\frac{c \log n}{\log(1/\varepsilon)} + O(\log n) + \log\log n\right) = n^{O_\varepsilon(1)}$$

time. □

6.3 Global Repositioning Algorithm for Insdel Errors

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

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

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

ALGORITHM 3: Global Repositioning Algorithm for Insertions and Deletions

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

Theorem 6.13. *Let d_i and d_r denote the number of symbols inserted into and deleted from the communication, respectively. The global repositioning algorithm formalized in Algorithm 3 guarantees a maximum misdecoding count of $(n + d_i - d_r)\beta + \frac{\varepsilon}{\beta}n$. More specifically, for $\beta = \sqrt{\varepsilon}$, the number of misdecodings is no larger than $3n\sqrt{\varepsilon}$ and running time will be $O(n^2/\sqrt{\varepsilon})$.*

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

Furthermore, we show that among all of successfully transmitted symbols, only $\frac{\varepsilon}{\beta}n$ can be decoded incorrectly. Take any matching M_i and consider all of the elements in M_i that correspond to a successfully transmitted but incorrectly decoded symbol. Let us indicate the elements of this set with $M_i' = \{(p_j, q_j)\}_j$. By definition, $S[p_j] = S'[q_j]$. Also, note that by the fact that $S'[q_j]$ is a successfully transmitted symbol, there is another symbol in S like $S[r_j]$ to which $S'[q_j]$ actually corresponds under the adversary's insertions and deletions. Therefore, for all j, we have that $S[r_j] = S'[q_j] = S[p_j]$ and $p_j \neq r_j$. This, along with the fact that both M_i' and $\{(r_j, q_j)\}_j$ are monotone matchings, gives that $M_i'' = \{(p_j, r_j)\}_j$ is a self-matching over S consisting only of bad matches. Thus, $|M_i''| \leq n\varepsilon$. This means that there are no more than $n\varepsilon$ misdecoded successfully transmitted symbols in M_i and, thus, no more than $\frac{n\varepsilon}{\beta}$ such misdecodings overall.

Adding up the two upper-bounds described above, one can bound the total number of misdecodings by $(n + d_i - d_r)\beta + \frac{\varepsilon}{\beta}n$. □

6.4 Global Repositioning Algorithm for Deletion Errors

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

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

Our simple streaming algorithm greedily matches S' to its left-most appearance in S as a subsequence. More precisely, the algorithm matches $S'[1]$ to $S[t_1']$ where t_1' is the smallest number where

$S[t'_1] = S'[1]$. Then, the algorithm matches $S'[2]$ to the smallest $t'_2 > t'_1$ where $S[t'_2] = S'[2]$ and construct the whole matching M' by repeating this procedure. Note that as there is a matching of size $|S'|$ between S and S', the size of the resulting matching M' will be $|S'|$ as well. We claim that the following holds for this algorithm:

Theorem 6.14. *Any ε-synchronization string along with the algorithm described in Section 6.4 form a linear-time streaming solution for deletion-only (n, δ)-indexing problem guaranteeing $\frac{\varepsilon}{1-\varepsilon} \cdot n\delta$ misdecodings.*

Proof. This algorithm clearly works in a streaming manner and runs in linear time. To analyze the performance, we make use of the fact that M and M' are both monotone matchings of size $|S'|$ between S and S'. Therefore, $\bar{M} = \{(t_1, t'_1), (t_2, t'_2), \ldots, (t_{n-d_r}, t'_{n-d_r})\}$ is a monotone matching between S and itself. Note that if $t_i \neq t'_i$, then the algorithm has incorrectly decoded the index symbol i. Let p be the number of all such symbols. Then matching $\bar{M}$ consists of $n - d_r - p$ good pairs and p bad pairs. Therefore, using Theorem 6.2, we have the following:

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

□

6.5 Global Repositioning Algorithm for Insertion Errors

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

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

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

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

Lemma 6.15. *Let $M_L = \{(1, l_1), (2, l_2), \ldots, (n, l_n)\}$ be the monotone matching between S and S', which yields the smallest lexicographical value for $l_1, \ldots, l_n$. We call M_L the* left-most *matching between S and S'. Similarly, let $M_R = \{(1, r_1), \ldots, (n, r_n)\}$ be the monotone matching for which $r_n, \ldots, r_1$ yields the largest possible lexicographical value. Then $S[i]$ is matched to $S'[t'_i]$ in all possible monotone matchings of size n between S and S' if and only if $(i, t'_i) \in M_R \cap M_L$.*

This lemma can be proved by a simple contradiction argument. Our algorithm starts by computing left-most and right-most monotone matchings between S and S' using the straightforward greedy algorithm introduced in Section 6.4 on (S, S') and their reversed versions. It then outputs the intersection of these two matchings as the answer.

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

elements in S', there has to be at least $2p - [(n + d_i) - (n - p)] = p - d_i$ elements in S' that are matched to different elements of S under M_L and M_R.

Consider the following monotone matching from S to itself.

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

Note that monotonicity follows the fact that both M_L and M_R are both monotone matchings between S and S'. We have shown that the size of the second set is at least $p - d_i$ and the size of the first set is by definition $n - p$. Also, all pairs in the first set are good pairs and all in the second one are bad pairs. Therefore, Theorem 6.2 implies that

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

which proves the efficiency claim and gives the following theorem.

THEOREM 6.16. *Any ε-synchronization string along with the algorithm described in Section 6.5 form a linear-time error-free solution for insertion-only (n, δ)-indexing problem guaranteeing $\frac{1}{1-\varepsilon} \cdot n\delta$ misdecodings.*

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

THEOREM 6.17. *Any ε-synchronization string along with the algorithm described in Section 6.5 form a linear-time error-free solution for deletion-only (n, δ)-indexing problem guaranteeing $\frac{\varepsilon}{1-\varepsilon} \cdot n\delta$ misdecodings.*

6.6 Linear-time Near-MDS Codes for Insertion-only and Deletion-only Settings

In the same manner as in Theorem 1.1, we can use error-free indexing solutions presented in Theorems 6.16 and 6.17 along with near-MDS (erasure) correcting codes to derive the following linear-time insertion-only or deletion-only errors.

THEOREM 6.18. *For any $\varepsilon > 0$ and $\delta \in (0, 1)$:*

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

6.7 Repositioning Using the Relative Suffix Pseudo-distance (RSPD)

In this section, we show how one can slightly improve the misdecoding guarantees in the results obtained in Section 5.2 by replacing RSD with a related notion of "distance" between two strings introduced in Reference [2]. We call this notion the *relative suffix pseudo-distance* or RSPD both to distinguish it from our RSD relative suffix distance and also because RSPD is not a metric—it is neither symmetric nor satisfies the triangle inequality.

Definition 6.19 (Relative Suffix Pseudo-distance (RSPD)). Given any two strings $c, \tilde{c} \in \Sigma^*$, the *relative suffix pseudo-distance* between c and $\tilde{c}$ is

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

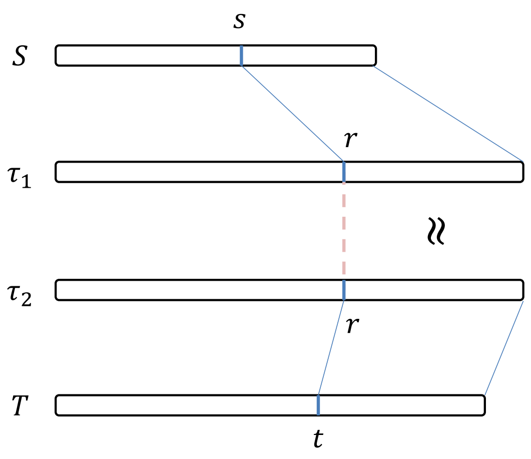


Fig. 2. Pictorial representation of the notation used in Lemma 6.21.

We remark that there is a loose resemblance between the RSPD and RSD notions. In the definition of the RSPD, the numerator of the maximization term counts the number of insertions and deletions that occur within a suffix of the communication given a fixed string matching between the two. The denominator holds the length of the part of the first string that lies within that suffix of the string matching. In other words, RSPD $(c, \tilde{c})$ refers to the largest density of insertions and deletions over all suffixes of the communication (described by a string matching), minimized over the choice of the string matching.

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

Lemma 6.20. *Let $S \in \Sigma^n$ be an ε-synchronization string and $\tilde{c} \in \Sigma^m$. Then, there exists at most one $c \in \bigcup_{i=1}^n \{S[1,i]\}$ such that* $\text{RSPD}(c, \tilde{c}) \leq 1 - \varepsilon$.

Before proceeding to the proof of Lemma 6.20, we prove the following lemma.

Lemma 6.21. *Let* $\text{RSPD}(S, T) \leq 1 - \varepsilon$*, then:*

(1) For every $1 \leq s \leq |S|$, there exists t such that $\text{ED}\left(S[s, |S|], T\left[t, |T|\right]\right) \leq (1-\varepsilon)(|S| - s + 1)$.
(2) For every $1 \leq t \leq |T|$, there exists s such that $\text{ED}\left(S[s, |S|], T\left[t, |T|\right]\right) \leq (1-\varepsilon)(|S| - s + 1)$.

Proof. Each part will be proved separately.

Part 1. Let τ be the string matching that attains the minimization in the definition of $\text{RSPD}(S, T)$. There exist some r such that $\text{del}(\tau_1\left[r, |\tau_1|\right]) = S[s, |S|]$. Note that $\text{del}(\tau_2[r, |\tau_2|])$ is a suffix of T. Therefore, there exists some t such that $T[t, |T|] = \text{del}(\tau_2[r, |\tau_2|])$. See Figure 2 for a pictorial representation. Now,

$$\begin{aligned}
\text{ED}(S[s, |S|], T[t, |T|]) &\leq sc(\text{del}(\tau_1\left[r, |\tau_1|\right])) + sc(\text{del}(\tau_2\left[r, |\tau_1|\right])) \\
&= \frac{sc(\text{del}(\tau_1\left[r, |\tau_1|\right])) + sc(\text{del}(\tau_2\left[r, |\tau_1|\right]))}{|\tau_1| - r + 1 - sc(\tau_1[r, |\tau_1|])} \cdot (|\tau_1| - r + 1 - sc(\tau_1[r, |\tau_1|])) \\
&\leq \text{RSPD}(S, T) \cdot (|S| - s + 1) \\
&\leq (1 - \varepsilon) \cdot (|S| - s + 1). \qquad (12)
\end{aligned}$$

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

```
Input: A received message $\tilde{c} \in \Sigma^m$ and an $\varepsilon$-synchronization string $S \in \Sigma^n$
1: $ans \leftarrow \emptyset$
2: for Any prefix $c$ of $S$ do
3:     $d[i][j][l] \leftarrow \min_{\substack{\tau:c(i)\to\tilde{c}(j)\\ sc(\tau_1)=l}} \max_{k=1}^{|\tau_1|} \frac{sc(\tau_1[k,|\tau_1|])+sc(\tau_2[k,|\tau_2|])}{|\tau_1|-k+1+sc(\tau_1[k,|\tau_1|])}$
4:     $\mathrm{RSPD}(c,\tilde{c}) \leftarrow \min_{l'=0}^{|\tilde{c}|} d[i][|\tilde{c}|][l']$
5:     if $\mathrm{RSPD}(c,\tilde{c}) \le 1-\varepsilon$ then
6:         $ans \leftarrow c$
Output: $ans$
```

Part 2. Similarly, let τ be the string matching yielding $\mathrm{RSPD}(S,T)$. There exists some r such that $\mathrm{del}(\tau_2\,[r,|\tau_2|]) = T[t,|T|]$. Now, $\mathrm{del}(\tau_1[r,|\tau_1|])$ is a suffix of S. Therefore, there exists some s such that $S[s,|S|] = \mathrm{del}(\tau_1[r,|\tau_1|])$. Now, all the steps we took to prove Equation (12) hold and the proof is complete. □

Proof of Lemma 6.20. For a contradiction, suppose that there exist $\tilde{c}$, l and l' such that $l < l'$, $\mathrm{RSPD}(S[1,l],\tilde{c}) \le 1-\varepsilon$ and $\mathrm{RSPD}(S[1,l'],\tilde{c}) \le 1-\varepsilon$. Now, using part 1 of Lemma 6.21, there exists k such that $\mathrm{ED}\,(S[l+1,l'],\tilde{c}[k,|\tilde{c}|]) \le (1-\varepsilon)(l'-l)$. Further, part 2 of Lemma 6.21 gives that there exist l'' such that $\mathrm{ED}\,(S[l''+1,l],\tilde{c}[k,|\tilde{c}|]) \le (1-\varepsilon)(l-l'')$. Hence,

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

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

In the same spirit as Section 5.2, one can develop a repositioning algorithm based on Lemma 6.20 that works as follows: upon arrival of each symbol, find a prefix of synchronization index string S that has the smallest RSPD to the string that is received so far (denoted by $\tilde{c}$). We call this algorithm the minimum RSPD decoding algorithm. Theorems 6.22 and 6.23 describe the computational complexity and misdecoding count of such repositioning algorithm.

Theorem 6.22. *Let $S \in \Sigma^n$ be an ε-synchronization string, and $\tilde{c} \in \Sigma^m$. Then Algorithm 4, given input S and $\tilde{c}$, either returns the unique prefix c of S such that* $\mathrm{RSPD}(c,\tilde{c}) \le 1-\varepsilon$ *or returns $\perp$ if no such prefix exists. Moreover, Algorithm 4 runs in $O(n^4)$ time. Therefore, using it over each received symbol to guess the position of all symbols of a communication, one derives a repositioning algorithm that runs in $O(n^5)$ time.*

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

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

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

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

This implies that

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

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

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

Theorem 6.23. *Suppose that S is an ε-synchronization string of length n over alphabet Σ that is sent over an insertion-deletion channel with c_i insertions and c_d deletions. Repositioning via using Algorithm 4 to guess the position of each received symbol results into no more than $\frac{c_i}{1-\varepsilon} + \frac{c_d\varepsilon}{1-\varepsilon}$ misdecodings.*

Proof. The proof of this theorem is similar to the proof of Theorem 5.11. Let prefix $S[1,i]$ be sent through the channel and $S_\tau[1,j]$ be received on the other end as the result of the adversary's set of actions $\tau = (\tau_1, \tau_2)$. Further, assume that $S_\tau[j]$ is successfully transmitted and is actually $S[i]$ sent by the other end. We first show that RSPD$(S[1,i], S'[1,j])$ is less than the relative suffix error density:

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

Now, using Lemma 5.15, we know that the relative suffix error density is smaller than $1-\varepsilon$ upon arrival of all but at most $\frac{c_i+c_d}{1-\varepsilon} - c_d$ of successfully transmitted symbols. Along with Lemma 6.20, this results into the conclusion that the minimum RSPD decoding algorithm guarantees no more than $\frac{c_i}{1-\varepsilon} + c_d(\frac{1}{1-\varepsilon} - 1)$ misdecodings. □

7 CONCLUSION AND FOLLOW-UP WORK

This article introduced synchronization strings as simple yet very powerful mathematical objects that are very helpful when dealing with synchronization errors in communications. In particular, one can use them to efficiently transform insertion and deletion errors into much simpler

Hamming-type errors in the context of error correcting errors. The article mainly focused on developing the theory of synchronization strings and its application in designing families of insdel codes with essentially optimal rate-distance trade-off in the finite-alphabet-size regime.

Ever since the preliminary publication of this article, several follow-up works have demonstrated further applications of synchronization strings in communication settings where coding against insertions and deletions are concerned.

A follow-up work by Haeupler et al. in 2017 [24] showed that the code construction methods presented in this article can be ported to the setting of binary codes to derive binary codes that tolerate a δ fraction of synchronization errors while achieving a rate of $1-O(\sqrt{\delta\log(1/\delta)})$. This improved upon the state-of-the-art insdel codes [13] with respect to the rate/distance trade-off. Two recent simultaneous works by Cheng et al. [5] and Haeupler [19] have further improved upon that by introducing efficient binary codes with rate $1 - O(\delta \log^2 \frac{1}{\delta})$ via providing deterministic document exchange protocols. Both of these works utilize mathematical structures that are inspired by structures introduced in this article such as ε-synchronization and ε-self-matching properties.

Further, Reference [24] shows that our synchronization string-based indexing method can be extended to fully simulate an ordinary substitution channel over a given insertion-deletion channel (i.e., without any delay for repositioning). This 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 interactive communications. This directly leads to new interactive coding schemes for the setting with synchronization errors introduced by Reference [2], including the first efficient scheme and the first scheme with good (and, in fact, likely near optimal) communication rate for small error fractions.

In this article, we provided a randomized polynomial time construction for ε-synchronization strings and a polynomial time deterministic construction for ε-self-matching strings. In Reference [22], several highly parallel, deterministic linear time constructions for ε-synchronization strings are given. In fact, these constructions provide highly explicit infinite ε-synchronization strings in which the ith symbol can be computed deterministically in $O(\log i)$ time. Reference [22] also gives strengthened versions of the ε-synchronization string property, which comes with very fast *local* decoding procedures and discuss several further applications.

Another follow-up work in 2019 [20] employs a similar indexing scheme with a specific pseudorandom string that enhances the global repositioning algorithm presented in Section 6.3 by decreasing its time complexity to near-linear time. This improves the decoding complexity of codes put forward in Theorem 1.1 to near-linear time.

Synchronization strings and our indexing methods also have shown to be useful to achieve families of list-decodable insdel codes with near-optimal rate-distance trade-off over large constant alphabets [23]. These synchronization string-based list-decodable insdel codes are used in a concatenation scheme in Reference [11] to provide optimally resilient list-decodable insertion-deletion codes.

Finally, in addition to applications, structural and combinatorial properties of synchronization strings were studied in a work by Cheng et al. [4].

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.

ACKNOWLEDGMENTS

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

REFERENCES

[1] Arturs Backurs and Piotr Indyk. 2018. Edit Distance cannot be computed in strongly subquadratic time (unless SETH is false). *SIAM J. Comput.* 47, 3 (2018), 1087–1097.

[2] Mark Braverman, Ran Gelles, Jieming Mao, and Rafail Ostrovsky. 2017. Coding for interactive communication correcting insertions and deletions. *IEEE Trans. Info. Theory* 63, 10 (2017), 6256–6270.

[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] Kuan Cheng, Bernhard Haeupler, Xin Li, Amirbehshad Shahrasbi, and Ke Wu. 2019. Synchronization strings: Highly efficient deterministic constructions over small alphabets. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA'19)*. 2185–2204.

[5] Kuan Cheng, Zhengzhong Jin, Xin Li, and Ke Wu. 2018. Deterministic document exchange protocols, and almost optimal binary codes for edit errors. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS'18).*

[6] Ran Gelles. 2017. Coding for interactive communication: A survey. *Found. Trends Theor. Comput. Sci.* 13, 1–2 (2017), 1–157.

[7] Ran Gelles and Bernhard Haeupler. 2017. Capacity of interactive communication over erasure channels and channels with feedback. *SIAM J. Comput.* 46, 4 (2017), 1449–1472.

[8] Mohsen Ghaffari and Bernhard Haeupler. 2014. Optimal Error Rates for Interactive Coding II: Efficiency and List Decoding. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS'14)*. 394–403.

[9] Mohsen Ghaffari, Bernhard Haeupler, and Madhu Sudan. 2014. Optimal Error Rates for Interactive Coding I: Adaptivity and other Settings. In *Proceedings of the ACM Symposium on Theory of Computing (STOC'14)*. 794–803.

[10] S. W. Golomb, J. Davey, I. Reed, H. Van Trees, and J. Stiffler. 1963. Synchronization. *IEEE Trans. Commun. Syst.* 11, 4 (1963), 481–491.

[11] Venkatesan Guruswami, Bernhard Haeupler, and Amirbehshad Shahrasbi. 2020. Optimally resilient codes for list-decoding from insertions and deletions. In *Proceedings of the ACM Symposium on Theory of Computing (STOC'20).*

[12] Venkatesan Guruswami and Piotr Indyk. 2005. Linear-time encodable/decodable codes with near-optimal rate. *IEEE Trans. Info. Theory* 51, 10 (2005), 3393–3400.

[13] Venkatesan Guruswami and Ray Li. 2016. Efficiently decodable insertion/deletion codes for high-noise and high-rate regimes. In *Proceedings of IEEE International Symposium on Information Theory (ISIT'16)*. 620–624.

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

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

[16] Venkatesan Guruswami and Carol Wang. 2017. Deletion codes in the high-noise and high-rate regimes. *IEEE Trans. Info. Theory* 63, 4 (2017), 1961–1970.

[17] Venkatesan Guruswami and Patrick Xia. 2015. Polar codes: Speed of polarization and polynomial gap to capacity. *IEEE Trans. Info. Theory* 61, 1 (2015), 3–16.

[18] Bernhard Haeupler. 2014. Interactive channel capacity revisited. In *Proceeding of the IEEE Symposium on Foundations of Computer Science (FOCS'14)*. 226–235.

[19] Bernhard Haeupler. 2019. Optimal document exchange and new codes for insertions and deletions. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS'19)*. 334–347.

[20] Bernhard Haeupler, Aviad Rubinstein, and Amirbehshad Shahrasbi. 2019. Near-linear time insertion-deletion codes and (1+ ε)-approximating edit distance via indexing. In *Proceedings of the ACM Symposium on Theory of Computing (STOC'19)*. 697–708.

[21] Bernhard Haeupler, Barna Saha, and Aravind Srinivasan. 2011. New constructive aspects of the lovász local lemma. *J. ACM* 58, 6 (2011), 28.

[22] Bernhard Haeupler and Amirbehshad Shahrasbi. 2018. Synchronization strings: Explicit constructions, local decoding, and applications. In *Proceedings of the ACM Symposium on Theory of Computing (STOC'18)*. 841–854.

[23] Bernhard Haeupler, Amirbehshad Shahrasbi, and Madhu Sudan. 2018. Synchronization strings: List decoding for insertions and deletions. In *Proceedings of the International Conference on Automata, Languages, and Programming (ICALP'18)*.

[24] Bernhard Haeupler, Amirbehshad Shahrasbi, and Ellen Vitercik. 2018. Synchronization Strings: Channel simulations and interactive coding for insertions and deletions. In *Proceedings of the International Conference on Automata, Languages, and Programming (ICALP'18)*.

[25] Gillat Kol and Ran Raz. 2013. Interactive channel capacity. In *Proceedings of the ACM Symposium on Theory of Computing (STOC'13)*. 715–724.

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

[27] S.-Y. R. Li, Raymond W. Yeung, and Ning Cai. 2003. Linear network coding. *IEEE Trans. Info. Theory* 49, 2 (2003), 371–381.

[28] Michael Luby. 2002. LT codes. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS'02)*. 271–282.

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

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

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

[32] Joseph Naor and Moni Naor. 1993. Small-bias probability spaces: Efficient constructions and applications. *SIAM J. Comput.* 22, 4 (1993), 838–856.

[33] Eran Rom and Amnon Ta-Shma. 2006. Improving the alphabet size in expander-based code constructions. *IEEE Trans. Info. Theory* 52, 8 (2006), 3695–3700.

[34] Leonard J. Schulman and David Zuckerman. 1999. Asymptotically good codes correcting insertions, deletions, and transpositions. *IEEE Trans. Info. Theory* 45, 7 (1999), 2552–2557.

[35] Neil J. A. Sloane. 2002. On single-deletion-correcting codes. In *Codes and Designs*, de Gruyter, Berlin, 273–291.

[36] Daniel A. Spielman. 1996. Linear-time encodable and decodable error-correcting codes. *IEEE Transactions on Information Theory (TransInf)* 42, 6 (1996), 1723–1731.

[37] A. Thue. 1977. Uber die gegenseitige lage gleicher teile gewisser zeichenreihen (1912). Selected mathematical papers of Axel Thue, Universitetsforlaget (1977).

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

Received March 2020; revised May 2021; accepted May 2021