Synchronization Strings: Explicit Constructions, Local Decoding, and Applications[*]

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

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

ABSTRACT

This paper gives new results for **synchronization strings**, a powerful combinatorial object introduced by [Haeupler, Shahrasbi; STOC'17] that allows to efficiently deal with insertions and deletions in various communication problems: (1) We give a **deterministic, linear time synchronization string construction**, improving over an $O(n^5)$ time randomized construction. Independently of this work, a deterministic $O(n \log^2 \log n)$ time construction was proposed by Cheng, Li, and Wu. (2) We give a **deterministic construction of an infinite synchronization string** which outputs the first n symbols in $O(n)$ time. Previously it was not known whether such a string was computable. (3) Both synchronization string constructions are **highly explicit**, i.e., the i^{th} symbol can be deterministically computed in $O(\log i)$ time. (4) This paper also introduces a generalized notion we call **long-distance synchronization strings**. Such strings **allow for local and very fast decoding**. In particular only $O(\log^3 n)$ time and access to logarithmically many symbols is required to decode any index.

The paper also provides several applications for these improved synchronization strings: (1) For any $\delta < 1$ and $\varepsilon > 0$ we provide an **insdel error correcting block code** with rate $1 - \delta - \varepsilon$ which can correct any $\delta/3$ fraction of insertion and deletion errors in $O(n \log^3 n)$ time. This **near linear computational efficiency** is surprising given that we do not even know how to compute the (edit) distance between the decoding input and output in subquadratic time. (2) We show that local decodability implies that error correcting codes constructed with long-distance synchronization strings can not only efficiently recover from δ fraction of insdel errors but, similar to [Schulman, Zuckerman; TransInf'99], also from any $O(\delta/\log n)$ fraction of **block transpositions and block replications**. These block corruptions allow arbitrarily long substrings to be swapped or replicated anywhere. (3) We show that highly explicitness and local decoding allow for **infinite channel simulations with exponentially smaller memory and decoding time requirements**. These simulations can then be used to give the first **near linear time interactive coding scheme for insdel errors**, similar to the result of [Brakerski, Naor; SODA'13] for Hamming errors.

CCS CONCEPTS

• **Mathematics of computing** → **Coding theory**; • **Theory of computation** → **Error-correcting codes**;

KEYWORDS

Synchronization Strings, Insertions and Deletions

ACM Reference Format:
Bernhard Haeupler and Amirbehshad Shahrasbi. 2018. Synchronization Strings: Explicit Constructions, Local Decoding, and Applications. In *Proceedings of 50th Annual ACM SIGACT Symposium on the Theory of Computing (STOC'18).* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3188745.3188940

1 INTRODUCTION

This paper gives new results for **ε-synchronization strings**, a powerful combinatorial object that can be used to effectively deal with insertions and deletions in various communication problems.

Synchronization strings are pseudo-random non-self-similar sequences of symbols over some finite alphabet that can be used to index a finite or infinite sequence of elements similar to the trivial indexing sequence $1, 2, 3, 4, \ldots, n$. In particular, if one first indexes a sequence of n elements with the trivial indexing sequence and then applies some k insertions or deletions of indexed elements one can still easily recover the original sequence of elements up to k half-errors, i.e., erasures or substitutions (where substitutions count twice). An ε-synchronization strings allows essentially the same up to an arbitrarily small error of εn half-errors but instead of having indexing symbols from a large alphabet of size n, which grows with the length of the sequence, a finite alphabet size of $\varepsilon^{-O(1)}$ suffices for ε-synchronization strings. Often this allows to efficiently transform insertion and deletion errors into ordinary Hamming errors which are much better understood and easier to handle.

One powerful application of synchronization strings is the design of efficient insdel error correcting codes (ECC), i.e., codes that can efficiently correct insertions and deletions. While codes for Hamming errors have been well understood making progress on insdel codes has been difficult [13, 16, 18, 26, 27, 33]. Synchronization strings solve this problem by transforming any regular error correcting block code C with a sufficiently large finite alphabet into an essentially equally efficient insdel code by simply indexing the symbols of C. This leads to the first insdel codes that approach the

[*] Supported in part by NSF grants CCF-1527110, CCF-1618280 and NSF CAREER award CCF-1750808. An extended version of this article is available at [21].

STOC'18, June 25–29, 2018, Los Angeles, CA, USA

ACM ISBN 978-1-4503-5559-9/18/06…$15.00
https://doi.org/10.1145/3188745.3188940

Singleton bound, i.e., for any $\delta < 1$ and $\varepsilon > 0$ one can get an insdel code with rate $1 - \delta - \varepsilon$ which, in quadratic time, recovers from any δ fraction of insertions or deletions. Further applications are given in [22, 23]. Most importantly, [23] introduces the notion of a channel simulation which allows one to use any insertion deletion channel like a black-box regular symbol corruption channel with a slightly increased error rate. This can be used to give the first computationally efficient interactive coding schemes for insdel errors and the first interactive coding scheme for insdel errors whose communication rate goes to one as the amount of noise goes to zero.

This paper provides drastically improved constructions of finite and infinite synchronization strings and a stronger synchronization string property which allows for decoding algorithms that are local and significantly faster. We furthermore give several applications for these results, including near linear time insertion-deletion codes, a near linear time coding scheme for interactive communication over insertion-deletion channels, exponentially better channel simulations in terms of time and memory, infinite channel simulations, and codes that can correct block transposition and block replication corruptions.

2 OUR RESULTS, STRUCTURE OF THIS PAPER, AND RELATED WORK

Next we give an overview of the main results and the overall structure of this paper. We also put our result in relation to related prior works.

2.1 Deterministic, Linear Time, Highly Explicit Construction of Infinite Synchronization Strings

In [20] the authors introduced synchronization strings and gave a $O(n^5)$ time randomized synchronization string construction. This construction could not be easily derandomized. In order to provide deterministic explicit constructions of insertion deletion block codes, [20] introduced a strictly weaker notion called self-matching strings, showed that these strings could be used for code constructions as well, and gave a deterministic $n^{O(1)}$ time self-matching string construction. Obtaining a deterministic construction for synchronization strings, however, was left open. [20] also showed the existence of infinite synchronization strings. This existence proof however is highly non-constructive. In fact, even the existence of a computable infinite synchronization string was left open; i.e., up to this paper there was no algorithm that would compute the i^{th} symbol of some infinite synchronization string in finite time.

In this paper, we give deterministic constructions of finite and infinite synchronization strings. Instead of going to a weaker notion, as done in [20], Section 4.1 introduces a stronger notion called long-distance synchronization strings. Interestingly, while the existence of these generalized synchronization strings can be shown with a similar Lovász local lemma based proof as for plain synchronization strings, this proof allows for an easier derandomization, which leads to a **deterministic polynomial time construction of (long-distance) synchronization strings**. Beyond this derandomization, the notion of long-distance synchronization strings turns out to be very useful and interesting in its own right, as will be shown later.

Next, two different boosting procedures, which make synchronization string constructions faster and more explicit, are given. The first boosting procedure, given in Section 4.4, leads to a **deterministic linear time synchronization string construction**. We remark that concurrently and independently Cheng, Li, and Wu obtained a deterministic $O(n \log^2 \log n)$ time synchronization string construction [9].

Our second boosting step, which is introduced in Section 4.3, makes our synchronization string construction **highly-explicit**, i.e., allows to compute any position of an n long synchronization string in time $O(\log n)$. This highly-explicitness is a property of crucial importance in most of our new applications.

Lastly, in Section 4.5 we give a simple transformation which allows us to use any construction for finite length synchronization strings and utilize it to give an construction of an **infinite synchronization string**. This transformation preserves highly-explicitness. Infinite synchronization strings are important for applications in which one has no a priori bound on the running time of a system, such as, streaming codes, channel simulations, and some interactive coding schemes. Overall we get the following simple to state theorem:

Theorem 2.1. *For any $0 < \varepsilon < 1$, there exists an infinite ε-synchronization string S over an alphabet of size $\varepsilon^{-O(1)}$ and a deterministic algorithm which for any i takes $O(\log i)$ time to compute $S[i, i+\log i]$, i.e., the i^{th} symbol of S (as well as the next $\log i$ symbols).*

Since any substring of an ε-synchronization string is also an ε-synchronization string itself this infinite synchronization string construction also implies a deterministic linear time construction of finite synchronization strings which is fully parallelizable. In particular, for any n there is a linear work parallel $\mathbf{NC}^1$ algorithm with depth $O(\log n)$ and $O(n/\log n)$ processors which computes the ε-synchronization string $S[1, n]$.

2.2 Long Distance Synchronization Strings and Fast Local Decoding

Section 5 shows that the long-distance property we introduced in Section 4.1, together with our highly explicit constructions from Section 4.3, allows the design of a much faster and highly local decoding procedure. In particular, to decode the index of an element in a stream that was indexed with a synchronization string it suffices to look at only $O(\log n)$ previously received symbols. The decoding of the index itself furthermore takes only $O(\log^3 n)$ time and can be done in a streaming fashion. This is significantly faster than the $O(n^3)$ streaming decoder or the $O(n^2)$ global decoder given in [20].

The paper furthermore gives several applications which demonstrate the power of these improved synchronization string constructions and the local decoding procedure.

2.3 Application: Codes Against Insdels, Block Transpositions and Replications

2.3.1 Near Linear Time Decodable Error Correcting Codes. Fast encoding and decoding procedures for error correcting codes have been important and influencial in both theory and practice. For regular error correcting block codes, the celebrated expander code framework given by Sipser and Spielman [32] and in Spielman's thesis [34] as well as later refinements by Alon, Edmonds, and Luby [1] as well as Guruswami and Indyk [14, 15] gave good ECCs with linear time encoding and decoding procedures. Very recently, a beautiful work by Hemenway, Ron-Zewi, and Wooters [24] achieved linear time decoding also for capacity achieving list decodable and locally list recoverable codes.

The synchronization string based insdel codes in [20] have linear encoding times but quadratic decoding times. As pointed out in [20], the latter seemed almost inherent to the harsher setting of insdel errors because *"in contrast to Hamming codes, even computing the distance between the received and the sent/decoded string is an edit distance computation. Edit distance computations in general do usually not run in sub-quadratic time, which is not surprising given the recent SETH-conditional lower bounds [2]".* Very surprisingly to us, our fast decoding procedure allows us to construct insdel codes with near linear decoding complexity:

Theorem 2.2. *For any $\delta < 1$ and $\varepsilon > 0$ there exists an* ***insdel error correcting block code*** *with rate $1 - \delta - \varepsilon$ that can correct from any $\delta/3$ fraction of insertions and deletions in $O(n \log^3 n)$ time. The encoding time is linear and the alphabet bit size is near linear in $\frac{1}{\delta+\varepsilon}$.*

Note that for any input string the decoder finds the codeword that is closest to it in edit distance, if a codeword with edit distance of at most $O(\delta n)$ exists. However, computing the distance between the input string and the codeword output by the decoder is an edit distance computation. Shockingly, even now, we do not know of any sub-quadratic algorithm that can compute or even crudely approximate this distance between input and output of our decoder, even though intuitively this seems to be much easier almost prerequisite step for the distance minimizing decoding problem itself. After all, decoding asks to find the closest (or a close) codeword to the input from an exponentially large set of codewords, which seems hard to do if one cannot even approximate the distance between the input and any particular codeword.

2.3.2 Application: High-Rate InsDel Codes that Efficiently Correct Block Transpositions and Replications. Section 6.2 gives another interesting application of our local decoding procedure. In particular, we show that local decodability directly implies that insdel ECCs constructed with our highly-explicit long-distance synchronization strings can not just efficiently recover from δ fraction of insdel errors but also from any $O(\delta/\log n)$ fraction of **block transpositions and block replications**. Block transpositions allow for arbitrarily long substrings to be swapped while a block replication allows for an arbitrarily long substring to be duplicated and inserted anywhere else. A similar result, albeit for block transpositions only, was shown by Schulman, Zuckerman [30] for the efficient constant distance constant rate insdel codes given by them. They also show that the $O(\delta/\log n)$ resilience against block errors is optimal up to constants.

2.4 Application: Exponentially More Efficient Infinite Channel Simulations

[23] introduced the powerful notion of a channel simulation. In particular, [23] showed that for any adversarial one-way or two-way insdel channel one can put a simple black-box at both ends such that to any two parties interacting with these black-boxes the behavior is indistinguishable from a much nicer Hamming channel which only introduces (a slightly larger fraction of) erasures and symbol corruptions. To achieve this these black-boxes were required to know a prior for how many steps T the channel would be used and required an amount of memory size that is linear in T. Furthermore, for each transmission at a time step t the receiving black-box would perform a $O(t^3)$ time computation. We show that using our locally decodable highly explicit long-distance synchronization strings can reduce both the memory requirement and the computation complexity exponentially. In particular each box is only required to have $O(\log t)$ bits of memory (which is optimal because at the very least it needs to store the current time) and any computation can be done in $O(\log^3 t)$ rounds. Furthermore due to our infinite synchronization string constructions the channel simulations black-boxes are not required to know anymore for how much time overall the channel will be used. These drastic improvements make channel simulations significantly more useful and indeed potentially quite practical.

2.5 Application: Near-Linear Time Interactive Coding Schemes for InsDel Errors

Interactive coding schemes, as introduced by Schulman [28, 29], allow to add redundancy to any interactive protocol between two parties in such a way that the resulting protocol becomes robust to noise in the communication. Interactive coding schemes that are robust to symbol corruptions have been intensely studied over the last few years [4, 5, 7, 10–12, 19, 25]. Similar to error correcting codes the main parameters for an interactive coding scheme is the fraction of errors it can tolerate[7, 10, 28, 29] its communication rate[19, 25] and its computational efficiency [4, 5, 11, 12]. In particular, Brakerski and Kalai [4] gave the first computationally efficient polynomial time interactive coding scheme. Brakerski and Naor [5] improved the complexity to near linear. Lastly, Ghaffari and Haeupler [12] gave a near-linear time interactive coding scheme that also achieved the optimal maximal robustness. More recently interactive coding schemes that are robust to insertions and deletions have been introduced by Braverman, Gelles, Mao, and Ostrovsky [6] subsequently Sherstov and Wu [31] gave a scheme with optimal error tolerance and Haeupler, Shahrasbi, and Vitercik [23] used channel simulations to give the first computationally efficient polynomial time interactive coding scheme for insdel errors. Our improved channel simulation can be used together with the coding scheme from [12] to directly get the first interactive coding scheme for insertions and deletions with a near linear time complexity - i.e., the equivalent of the result of Brakerski and Naor [5] but for insertions and deletions.

3 DEFINITIONS AND PRELIMINARIES

In this section, we provide the notation and definitions we will use throughout the rest of the paper. We also briefly review key definitions and techniques from [20, 23].

3.1 String Notation

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 we define S^k to equal k copies of S concatenated together. For $i, j \in \{1, \dots, n\}$, we denote the substring of S from the i^{th} index through and including the j^{th} index as $S[i, j]$. Such a consecutive substring is also called a *factor* of S. For $i < 1$ we define $S[i, j] = \perp^{-i+1} \cdot S[1, j]$ where $\perp$ is a special symbol not contained in Σ. We refer to the substring from the i^{th} index through, but not including, the j^{th} index as $S[i, j)$. The substrings $S(i, j]$ and $S(i, j)$ are similarly defined. $S[i]$ denotes the i^{th} symbol of S and $|S| = n$ is the length of S. Occasionally, the alphabets we use are the cross-product of several alphabets, i.e. $\Sigma = \Sigma_1 \times \cdots \times \Sigma_n$. If T is a string over Σ, then we write $T[i] = [a_1, \dots, a_n]$, where $a_i \in \Sigma_i$. Finally, symbol by symbol concatenation of two strings S and T of similar length is $[(S_1, T_1), (S_2, T_2), \cdots]$.

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

Definition 3.1 (Edit distance). The *edit distance* $\mathrm{ED}(c, c')$ between two strings $c, c' \in \Sigma^*$ is the minimum number of insertions and deletions required to transform c into c'.

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

Definition 3.2 (Relative Suffix Distance). For any two strings $S, S' \in \Sigma^*$ we define their relative suffix distance RSD as follows:

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

Lemma 3.3. *For any strings S_1, S_2, S_3 we have*

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

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

3.2 Synchronization Strings

We now recall synchronization string based techniques and relevant lemmas from [20, 23] which we will be of use here. In short, synchronization strings allow communicating parties to protect against synchronization errors by indexing their messages without blowing up the communication rate. The general idea of coding schemes introduced and utilized in [20, 23], is to index any communicated symbol in the sender side and then *guess* the actual position of received symbols on the other end using the attached indices.

A straightforward candidate for such technique is to attach $1, \cdots, n$ to communicated symbols where n indicates the rounds of communication. However, this trivial indexing scheme would not lead to an efficient solution as it requires assigning a $\log n$-sized space to indexing symbols. This shortcoming accentuates a natural trade-off between the size of the alphabet among which indexing symbols are chosen and the accuracy of the guessing procedure on the receiver side.

Haeupler and Shahrasbi [20] introduce ε-synchronization strings as well-fitting candidates for this matter. This family of strings, parametrized by ε, are over alphabets of constant size in terms of communication length n and dependent merely on parameter ε. ε-synchronization strings can convert any adversarial k synchronization errors into hamming-type errors. The extent of disparity between the number translated hamming-type errors and k can be controlled by parameter ε.

Imagine Alice and Bob as two parties communicating over a channel suffering from up to δ-fraction of adversarial insertions and deletions. Suppose Alice sends a string S of length n to Bob. On the other end of the communication, Bob will receive a distorted version of S as adversary might have inserted or deleted a number of symbols. A symbol which is sent by Alice and is received by Bob without being deleted by the adversary is called a *successfully transmitted* symbol.

Assume that Alice and Bob both know string S a priori. Bob runs an algorithm to determine the actual index of each of the symbols he receives, in other words, to guess which element of S they correspond to. Such algorithm has to return an number in $[1, n]$ or "I don't know" for any symbol of S_τ. We call such an algorithm an *(n, δ)-indexing algorithm.*

Ideally, a indexing algorithm is supposed to correctly figure out the indices of as many successfully transmitted symbols as possible. The measure of *misdecodings* has been introduced in [20] to evaluate the quality of a (n, δ)-indexing algorithm as the number of successfully transmitted symbols that an algorithm might not decoded correctly. An indexing algorithm is called to be *streaming* if its output for a particular received symbol depends only on the symbols that have been received before it.

Haeupler and Shahrasbi [20] discuss ε-synchronization strings along with several decoding techniques for them.

Definition 3.4 (ε-Synchronization String). String $S \in \Sigma^n$ is an ε-synchronization string if for every $1 \le i < j < k \le n + 1$ we have that $ED(S[i, j), S[j, k)) > (1 - \varepsilon)(k - i)$. We call the set of prefixes of such a string an ε-synchronization code.

We will make use of the global decoding algorithm from [20] described as follows.

Theorem 3.5 (Theorems and 6.14 from [20]). *There is a decoding algorithm for an ε-synchronization string of length n which guarantees decoding with up to $O(n\sqrt{\varepsilon})$ misdecodings and runs in $O(n^2/\sqrt{\varepsilon})$ time.*

Theorem 3.6 (Theorem 4.1 from [20]). *Given a synchronization string S over alphabet Σ_S with an (efficient) decoding algorithm $\mathcal{D}_S$ guaranteeing at most k misdecodings and decoding complexity $T_{\mathcal{D}_S}(n)$ and an (efficient) ECC C over alphabet Σ_C with rate R_C, encoding complexity $T_{\mathcal{E}_C}$, and decoding complexity $T_{\mathcal{D}_C}$ that corrects up to $n\delta + 2k$ half-errors, one obtains an insdel code that can be (efficiently) decoded from up to $n\delta$ insertions and deletions. The rate of this code is at least $\frac{R_C}{1+\log|\Sigma_S|/\log|\Sigma_C|}$ The encoding complexity remains*

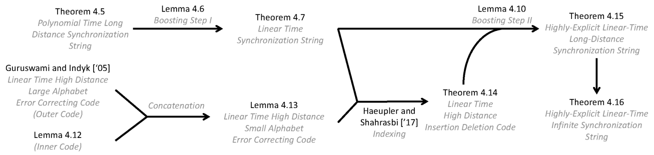
Theorem 4.5
Polynomial Time Long Distance Synchronization String
Lemma 4.6
Boosting Step I
Theorem 4.7
Linear Time Synchronization String
Lemma 4.10
Boosting Step II
Theorem 4.15
Highly-Explicit Linear-Time Long-Distance Synchronization String
Guruswami and Indyk ['05]
Linear Time High Distance Large Alphabet Error Correcting Code (Outer Code)
Lemma 4.12
(Inner Code)
Concatenation
Lemma 4.13
Linear Time High Distance Small Alphabet Error Correcting Code
Haeupler and Shahrasbi ['17]
Indexing
Theorem 4.14
Linear Time High Distance Insertion Deletion Code
Theorem 4.16
Highly-Explicit Linear-Time Infinite Synchronization String

Figure 1: Schematic flow of Theorems and Lemmas of Section 4

$T_{\mathcal{E}_C}$, the decoding complexity is $T_{\mathcal{D}_C} + T_{\mathcal{D}_S}(n)$ and the preprocessing complexity of constructing the code is the complexity of constructing C and S.

4 HIGHLY EXPLICIT CONSTRUCTIONS OF LONG-DISTANCE AND INFINITE ε-SYNCHRONIZATION STRINGS

We start this section by introducing a generalized notion of synchronization strings in Section 4.1 and then provide a deterministic efficient construction for them in Section 4.2. In Section 4.3, we provide a boosting step which speeds up the construction to linear time in Theorem 4.7. In Section 4.4, we use the linear time construction to obtain a linear-time high-distance insdel code (Theorem 4.14) and then use another boosting step to obtain a highly-explicit linear-time construction for long-distance synchronization strings in Theorem 4.15. We provide similar construction for infinite synchronization strings in Section 4.5. A pictorial representation of the flow of theorems and lemmas in this section can be found in Figure 1.

4.1 Long-Distance Synchronization Strings

The existence of synchronization strings is proven in [20] using an argument based on Lovász local lemma. This lead to an efficient randomized construction for synchronization strings which cannot be easily derandomized. Instead, the authors introduced the weaker notion of self-matching strings and gave a deterministic construction for them. Interestingly, in this paper we introduce a revised notion, denoted by *$f(l)$-distance ε-synchronization strings*, which generalizes ε-synchronization strings and allows for a deterministic construction.

Note that the synchronization string property poses a requirement on the edit distance of neighboring substrings. $f(l)$-distance ε-synchronization string property extends this requirement to any pair of intervals that are nearby. More formally, any two intervals of aggregated length l that are of distance $f(l)$ or less have to satisfy the edit distance property in this generalized notion.

Definition 4.1 ($f(l)$-distance ε-synchronization string). String $S \in \Sigma^n$ is an $f(l)$-distance ε-synchronization string if for every $1 \le i < j \le i' < j' \le n+1$ we have that $ED(S[i,j), S[i',j')) > (1-\varepsilon)l$ if $i' - j \le f(l)$ where $l = j + j' - i - i'$.

It is noteworthy to mention that the constant function $f(l) = 0$ gives the original ε-synchronization strings. Haeupler and Shahrasbi [20] have studied the existence and construction of synchronization strings for this case. In particular, they have shown that arbitrarily long ε-synchronization strings exist over an alphabet that is polynomially large in terms of ε^{-1}. Besides $f(l) = 0$, there are several other functions that might be of interest in this context.

One can show that, as we do in the full version of this paper, that for any polynomial function $f(l)$, arbitrarily long $f(l)$-distance ε-synchronization strings exist over alphabet sizes that are polynomially large in terms of ε^{-1}. Also, for exponential functions, these strings exist over exponentially large alphabets in terms of ε^{-1} but not over sub-exponential alphabet sizes. Finally, if function f is super-exponential, $f(l)$-distance ε-synchronization strings do not exist over any constant size alphabet. The similar question of constructing infinite binary strings that avoid identical substrings of length n with exponential distance in terms of n have been studied by Beck [3].

While studying existence, construction, and alphabet sizes of $f(l)$-distance ε-synchronization strings might be of interest by its own, we will show that having synchronization string edit distance guarantee for pairs of intervals that are exponentially far in terms of their aggregated length is of significant interest as it leads to improvements over applications of ordinary synchronization strings described in [20, 23] from several aspects. Even though distance function $f(l) = c^l$ provides such property, throughout the rest of this paper, we will focus on a variant of it, i.e., $f(l) = n \cdot \mathbb{1}_{l > c\log n}$ which allows polynomial-sized alphabet. $\mathbb{1}_{l > c\log n}$ is the indicator function for $l > c\log n$, i.e., one if $l > c\log n$ and zero otherwise

To compare distance functions $f(l) = c^l$ and $f(l) = n \cdot \mathbb{1}_{l > c\log n}$, note that the first one allows intervals to be exponentially far away in their total length. In particular, intervals of length $l > c\log n$ or larger can be arbitrarily far away. The second function only asks for the guarantee over large intervals and does not strengthen the ε-synchronization property for smaller intervals. We refer to the later as *c-long-distance ε-synchronization string* property.

Definition 4.2 (c-long-distance ε-synchronization strings). We call $n \cdot \mathbb{1}_{l > c\log n}$-distance ε-synchronization strings *c-long-distance ε-synchronization strings.*

4.2 Polynomial Time Construction of Long-Distance Synchronization Strings

An LLL-based proof for existence of ordinary synchronization strings has been provided by [20]. Here we provide a similar technique along with the deterministic algorithm for Lovász local lemma from Chandrasekaran et al.[8] to prove the existence and give a deterministic polynomial-time construction of strings that satisfy this quality over an alphabet of size $\varepsilon^{-O(1)}$.

Before giving this proof right away, we first show a property of these strings which allows us to simplify the proof and, more importantly, get a deterministic algorithm using deterministic algorithms for Lovász local lemma from Chandrasekaran et al.[8].

Lemma 4.3. *If S is a string and there are two intervals $i_1 < j_1 \le i_2 < j_2$ of total length $l = j_1 - i_1 + j_2 - i_2$ and* $\mathrm{ED}(S[i_1, j_1), S[i_2, j_2)) \le (1-\varepsilon)l$ *then there also exists intervals $i_1 \le i_1' < j_1' \le i_2' < j_2' \le j_2$ of total length $l' \in \{\lceil l/2\rceil - 1, \lceil l/2\rceil, \lceil l/2\rceil + 1\}$ with* $\mathrm{ED}(S[i_1', j_1'), S[i_2', j_2'))$ $\le (1-\varepsilon)l'$.

Proof. As $\mathrm{ED}(S[i_1, j_1), [i_2, j_2)) \le (1-\varepsilon)l$, there has to be a common subsequence of length $m \ge \frac{\varepsilon l}{2}$ between $S[i_1, j_1)$ and $S[i_2, j_2)$

locating at indices $a_1 < a_2 < \cdots < a_m$ and $b_1 < b_2 < \cdots < b_m$ respectively. We call $M = \{(a_1, b_1), \cdots, (a_m, b_m)\}$ a monotone matching from $S[i_1, j_1)$ to $S[i_2, j_2)$. Let $1 \le i \le m$ be the largest number such that $|S[i_1, a_i]| + |S[i_2, b_i]| \le \lceil l/2 \rceil$. It is easy to verify that there are integers $a_i < k_1 \le a_{i+1}$ and $b_i < k_2 \le b_{i+1}$ such that $|S[i_1, k_1)| + |S[i_2, k_2)| \in \{\lceil l/2 \rceil - 1, \lceil l/2 \rceil\}$.

Therefore, we can split the pair of intervals $(S[i_1, j_1), S[i_2, j_2))$ into two pairs of intervals $(S[i_1, k_1), S[i_2, k_2))$ and $(S[k_1, j_1), S[k_2, j_2))$ such that each pair of the matching M falls into exactly one of these pairs. Hence, in at least one of those pairs, the size of the matching is larger than $\frac{\varepsilon}{2}$ times the total length. This gives that the edit distance of those pairs is less than $1 - \varepsilon$ and finishes the proof. □

Lemma 4.3 shows that if there is a pair of intervals of total length l that have small relative edit distance, we can find a pair of intervals of size $\{\lceil l/2 \rceil - 1, \lceil l/2 \rceil, \lceil l/2 \rceil + 1\}$ which have small relative edit distance as well. Now, let us consider a string S with a pair of intervals that violate the c-long distance ε-synchronization property. If the total length of the intervals exceed $2c \log n$, using Lemma 4.3 we can find another pair of intervals of almost half the total length which still violate the c-long distance ε-synchronization property. Note that as their total length is longer than $c \log n$, we do not worry about the distance of those intervals. Repeating this procedure, we can eventually find a pair of intervals of a total length between $c \log n$ and $2c \log n$ that violate the c-long distance ε-synchronization property. More formally, we can derive the following statement by Lemma 4.3.

Corollary 4.4. *If S is a string which satisfies the c-long-distance ε-synchronization property for any two non-adjacent intervals of total length $2c \log n$ or less, then it satisfies the property for all pairs of non-adjacent intervals.*

Proof. Suppose, for the sake of contradiction, that there exist two intervals of total length $2 \log_c n$ or more that violate the c-long-distance ε-synchronization property. Let $[i_1, j_1)$ and $[i_2, j_2)$ where $i_1 < j_1 \le i_2 < j_2$ be two intervals of the smallest total length $l = j_1 - i_1 + j_2 - i_2$ larger than $2 \log_c n$ (breaking ties arbitrarely) for which $\mathrm{ED}(S[i_1, j_1), [i_2, j_2)) \le (1-\varepsilon)l$. By Lemma 4.3 there exists two intervals $[i'_1, j'_1)$ and $[i'_2, j'_2)$ where $i'_1 < j'_1 \le i'_2 < j'_2$ of total length $l' \in [l/2, l)$ with $\mathrm{ED}(S[i'_1, j'_1), [i'_2, j'_2)) \le (1 - \varepsilon)l$. If $l' \le 2 \log_c n$, the assumption of c-long-distance ε-synchronization property holding for intervals of length $2 \log_c n$ or less is contradicted. Unless, $l' > 2 \log_c n$ that contradicts the minimality of our choice of l. □

Theorem 4.5. *For any $0 < \varepsilon < 1$ and every n there is a deterministic $n^{O(1)}$ time algorithm for computing a $c = O(1/\varepsilon)$-long-distance ε-synchronization string over an alphabet of size $O(\varepsilon^{-4})$.*

Proof. To prove this, we will make use of the Lovász local lemma and deterministic algorithms proposed for it in [8]. We generate a random string R over an alphabet of size $|\Sigma| = O(\varepsilon^{-2})$ and define bad event B_{i_1, l_1, i_2, l_2} as the event of intervals $[i_1, i_1 + l_1)$ and $[i_2, i_2 + l_2)$ violating the $O(1/\varepsilon)$-long-distance synchronization string property over intervals of total length $2/\varepsilon^2$ or more. In other words, B_{i_1, l_1, i_2, l_2} occurs if and only if $\mathrm{ED}(R[i_1, i_1 + l_1), R[i_2, i_2 + l_2)) \le (1 - \varepsilon)(l_1 + l_2)$. Note that by the definition of c-long-distance ε-synchronization strings, B_{i_1, l_1, i_2, l_2} is defined for (i_1, l_1, i_2, l_2)s where either $l_1 + l_2 \ge c \log n$ and $i_1 + l_1 \le i_2$ or $2/\varepsilon^2 < l_1 + l_2 < c \log n$ and $i_2 = i_1 + l_1$. We aim to show that for large enough n, with non-zero probability, none of these bad events happen. This will prove the existence of a string that satisfies $c = O(1/\varepsilon)$-long-distance ε-synchronization strings for all pairs of intervals that are of total length $2/\varepsilon^2$ or more. To turn this string into a $c = O(1/\varepsilon)$-long-distance ε-synchronization strings, we simply concatenate it with a string consisting of repetitions of $1, \cdots, 2\varepsilon^{-2}$, i.e., $1, 2, \cdots, 2\varepsilon^{-2}, 1, 2, \cdots, 2\varepsilon^{-2}, \cdots$. This string will take care of the edit distance requirement for neighboring intervals with total length smaller than $2\varepsilon^{-2}$.

Note that using Lemma 4.3, by a similar argument as in Claim 4.4, we only need to consider bad events where $l_1 + l_2 \le 2c \log n$. As the first step, note that B_{i_1, l_1, i_2, l_2} happens only if there is a common subsequence of length $\varepsilon(l_1 + l_2)/2$ or more between $R[i_1, i_1 + l_1)$ and $R[i_2, i_2 + l_2)$. Hence, the union bound gives that

$$\begin{aligned}\Pr\left\{B_{i_1, l_1, i_2, l_2}\right\} &\le \binom{l_1}{\varepsilon(l_1 + l_2)/2}\binom{l_1}{\varepsilon(l_1 + l_2)/2}|\Sigma|^{-\frac{\varepsilon(l_1+l_2)}{2}} \\ &\le \left(\frac{2e\sqrt{l_1 l_2}}{\varepsilon(l_1 + l_2)\sqrt{|\Sigma|}}\right)^{\varepsilon(l_1+l_2)} \le \left(\frac{e}{\varepsilon\sqrt{|\Sigma|}}\right)^{\varepsilon l}\end{aligned}$$

where $l = l_1 + l_2$. In order to apply LLL, we need to find real numbers $x_{i_1, l_1, i_2, l_2} \in [0, 1]$ such that for any B_{i_1, l_1, i_2, l_2}

$$\Pr\{B_{i_1, l_1, i_2, l_2}\} \le x_{i_1, l_1, i_2, l_2} \cdot \prod_{[S[i_1, i_1+l_1) \cup S[i_2, i_2+l_2)] \cap [S[i'_1, i'_1+l'_1) \cup S[i'_2, i'_2+l'_2)] \neq \emptyset} (1 - x_{i'_1, l'_1, i'_2, l'_2}) \quad (1)$$

We eventually want to show that our LLL argument satisfies the conditions required for polynomial-time deterministic algorithmic LLL specified in [8]. Namely, it suffices to certify two other properties in addition to (1). The first additional requirement is to have each bad event in LLL depend on up to logarithmically many variables and the second is to have (1) hold with a constant exponential slack. The former is clearly true as our bad events consist of pairs of intervals each of which is of a length between $c \log n$ and $2c \log n$. To have the second requirement, instead of (1) we find $x_{i_1, l_1, i_2, l_2} \in [0, 1]$ that satisfy the following stronger property.

$$\Pr\{B_{i_1, l_1, i_2, l_2}\} \le \left[x_{i_1, l_1, i_2, l_2} \cdot \prod_{[S[i_1, i_1+l_1) \cup S[i_2, i_2+l_2)] \cap [S[i'_1, i'_1+l'_1) \cup S[i'_2, i'_2+l'_2)] \neq \emptyset} (1 - x_{i'_1, l'_1, i'_2, l'_2})\right]^{1.01} \quad (2)$$

Any small constant can be used as slack. We pick 1.01 for the sake of simplicity. We propose $x_{i_1, l_1, i_2, l_2} = 2^{-\varepsilon(l_1+l_2)}$. It can be shown that for sufficiently small ε, $c = 2/\varepsilon$, and some $|\Sigma| = O(\varepsilon^{-2})$, this choice of x satisfies (2) and, therefore, completes the proof. The details of this proof are available in the extended version of this paper [21]. □

4.3 Boosting I: Linear Time Construction of Synchronization Strings

Next, we provide a simple boosting step which allows us to polynomially speed up any ε-synchronization string construction. Essentially, we propose a way to construct an $O(\varepsilon)$-synchronization string of length $O_\varepsilon(n^2)$ having an ε-synchronization string of length n.

Lemma 4.6. *Fix an even $n \in \mathbb{N}$ and $\gamma > 0$ such that $\gamma n \in \mathbb{N}$. Suppose $S \in \Sigma^n$ is an ε-synchronization string. The string $S' \in \Sigma'^{\gamma n^2}$ with $\Sigma' = \Sigma^3$ and*

$$S'[i] = \left(S[i \bmod n], S[(i + n/2) \bmod n], S\left[\left\lceil \frac{i}{\gamma n} \right\rceil\right]\right)$$

is an $(\varepsilon + 6\gamma)$-synchronization string of length γn^2.

Proof. Intervals of length at most $n/2$ lay completely within a copy of S and thus have the ε-synchronization property. For intervals of size l larger than $n/2$ we look at the synchronization string which is blown up by repeating each symbol γn times. Ensuring that both sub-intervals contain complete blocks changes the edit distance by at most $3\gamma n$ and thus by at most $6\gamma l$. Once only complete blocks are contained we use the observation that the longest common subsequence of any two strings becomes exactly a factor k larger if each symbols is repeated k times in each string. This means that the relative edit distance does not change and is thus at least ε. Overall this results in the $(\varepsilon + 6\gamma)$-synchronization string property to hold for large intervals in S'. □

We use this step to speed up the polynomial time deterministic ε-synchronization string construction in Theorem 4.5 to linear time.

Theorem 4.7. *There exists an algorithm that, for any $0 < \varepsilon < 1$, constructs an ε-synchronization string of length n over an alphabet of size $\varepsilon^{-O(1)}$ in $O(n)$ time.*

Proof. Note that if one takes an ε'-synchronization strings of length n' and applies the boosting step in Theorem 4.6 k times with parameter γ, he would obtain a $(\varepsilon' + 6k\gamma)$-synchronization string of length $\gamma^{2^k-1}n^{2^k}$.

For any $0 < \varepsilon < 1$, Theorem 4.5 gives a deterministic algorithm for constructing an ε-synchronization string over an alphabet $O(\varepsilon^{-4})$ that takes $O(n^T)$ time for some constant T independent of ε and n. We use the algorithm in Theorem 4.5 to construct an $\varepsilon' = \frac{\varepsilon}{2}$ synchronization string of length $n' = \frac{n^{1/T}}{\gamma}$ for $\gamma = \frac{\varepsilon}{12 \log T}$ over an alphabet of size $O(\varepsilon^{-4})$ in $O(n'^T) = O(n)$ time. Then, we apply boosting step I $k = \log T$ times with $\gamma = \frac{\varepsilon}{12 \log T}$ to get an $(\varepsilon' + 6\gamma \log T = \varepsilon)$-synchronization string of length $\gamma^{T-1}n'^T \geq n$. As boosting step have been employed constant times, the eventual alphabet size will be $\varepsilon^{-O(1)}$ and the run time is $O(n)$. □

4.4 Boosting II: Explicit Constructions for Long-Distance Synchronization Strings

We start this section by a discussion of *explicitness* quality of synchronization string constructions. In addition to the time complexity of synchronization strings' constructions, an important quality of a construction that we take into consideration for applications that we will discuss later is explicitness or, in other words, how fast one can calculate a particular symbol of a synchronization string.

Definition 4.8 ($T(n)$-explicit construction). If a synchronization string construction algorithm can compute ith index of the string it is supposed to find, i.e., $S[i]$, in $T(n)$ we call it an $T(n)$-*explicit* algorithm.

We are particularly interested in cases where $T(n)$ is polylogarithmically large in terms of n. For such $T(n)$, a $T(n)$-explicit construction implies a near-linear construction of the entire string as one can simply compute the string by finding out symbols one by one in $n \cdot T(n)$ overall time. We use the term *highly-explicit* to refer to $O(\log n)$-explicit constructions.

We now introduce a boosting step in Lemma 4.10 that will lead to explicit constructions of (long-distance) synchronization strings. Lemma 4.10 shows that, using a high-distance insertion-deletion code, one can construct strings that satisfy the requirement of long-distance synchronization strings for every pair of substrings that are of total length $\Omega_\varepsilon(\log n)$ or more. Having such a string, one can construct a $O_\varepsilon(1)$-long-distance ε-synchronization string by simply concatenating the outcome of Lemma 4.10 with repetitions of an $O_\varepsilon(\log n)$-long ε-synchronization string.

This boosting step is deeply connected to our new definition of long-distance ε-synchronization strings. In particular, we observe the following interesting connection between insertion-deletion codes and long-distance ε-synchronization strings.

Lemma 4.9. *If S is a c-long-distance ε-synchronization string over an alphabet of size q where $c = \Theta(1)$ then $C = \{S(i \cdot c \log n, (i+1) \cdot c \log n] | 0 \leq i < \frac{n}{c \log n} - 1\}$ is an insdel error correcting code with minimum distance at least $1 - \varepsilon$ and constant rate $\Omega_q(1)$. Further, if any substring $S[i, i + \log n]$ is computable in $O(\log n)$ time, C has a linear encoding time.*

Proof. The distance follows from the definition of long-distance ε-synchronization strings. The rate follows because the rate R is equal to $R = \frac{\log |C|}{c \log n \log q} = \frac{\log \frac{n}{c \log n}}{O_q(\log n)} = \Omega_q(1)$. Finally, since $|S(i \cdot c \log n, (i+1) \cdot c \log n]| = c \log n$, one can compute $S(i \cdot c \log n, (i+1) \cdot c \log n]$ in linear time in terms of its length. □

Our boosting step is mainly built on the converse of this observation.

Lemma 4.10. *Suppose C is a block insdel code over alphabet of size q, block length N, distance $1 - \varepsilon$ and rate R and let S be a string obtained by attaching all codewords back to back in any order. Then, for $\varepsilon' = 4\varepsilon$, S is a string of length $n = q^{R \cdot N} \cdot N$ which satisfies the long-distance ε'-synchronization property for any pair of intervals of aggregated length $\frac{4}{\varepsilon}N \leq \frac{4}{\varepsilon \log q}(\log n - \log R)$ or more. Further, if C is linear-time encodable, S has a highly explicit construction.*

Proof. The length of S follows from the definition of rate. Moreover, the highly explicitness follows from the fact that every substring of S of length $\log n$ may include parts of $\frac{1}{\varepsilon \log q} + 1$ codewords each of which can be computed in linear time in terms of their length. Therefore, any substring $S[i, i + \log n]$ can be constructed in $O\left(\max\left\{\frac{\log n}{\varepsilon \log q}, \log n\right\}\right) = O_{\varepsilon,q}(\log n)$. To prove the long distance property, we have to show that for every four indices $i_1 < j_1 \leq i_2 < j_2$ where $j_1 + j_2 - i_1 - i_2 \geq \frac{4N}{\varepsilon}$, we have

$$\text{ED}(S[i_1, j_1), S[i_2, j_2)) \geq (1 - 4\varepsilon)(j_1 + j_2 - i_1 - i_2). \tag{3}$$

Assume that $S[i_1, j_1)$ contains a total of p complete blocks of C and $S[i_2, j_2)$ contains q complete blocks of C. Let $S[i'_1, j'_1)$ and $S[i'_2, j'_2)$ be the strings obtained be throwing the partial blocks away from $S[i_1, j_1)$ and $S[i_2, j_2)$. Note that the overall length of the partial blocks in $S[i_1, j_1)$ and $S[i_2, j_2)$ is less than $4N$, which is at most an ε-fraction of $S[i_1, j_1) \cup S[i_2, j_2)$, since $\frac{4N}{4N/\varepsilon} < \varepsilon$.

Assume by contradiction that $\mathrm{ED}(S[i_1, j_1), S[i_2, j_2)) < (1-4\varepsilon)(j_1 + j_2 - i_1 - i_2)$. Since edit distance preserves the triangle inequality, we have that

$$\begin{aligned} ED\left(S[i'_1, j'_1), S[i'_2, j'_2)\right) &\leq ED\left(S[i_1, j_1), S[i_2, j_2)\right) + |S[i_1, i'_1)| + \\ &\quad |S[j'_1, j_1)| + |S[i_2, i'_2)| + |S[j'_2, j_2)| \\ &\leq (1 - 4\varepsilon + \varepsilon)\left(j_1 + j_2 - i_1 - i_2\right) \\ &< \left(\frac{1-3\varepsilon}{1-\varepsilon}\right)\left((j'_1 - i'_1) + (j'_2 - i'_2)\right). \end{aligned}$$

This means that the longest common subsequence of $S[i'_1, j'_1)$ and $S[i'_2, j'_2)$ has length of at least

$$\frac{1}{2}\left[\left(|S[i'_1, j'_1)| + |S[i'_2, j'_2)|\right)\left(1 - \frac{1-3\varepsilon}{1-\varepsilon}\right)\right],$$

which means that there exists a monotonically increasing matching between $S[i'_1, j'_1)$ and $S[i'_2, j'_2)$ of the same size. Since the matching is monotone, there can be at most $p + q$ pairs of error-correcting code blocks having edges to each other. The Pigeonhole Principle implies that there are two error-correcting code blocks B_1 and B_2 such that the number of edges between them is at least

$$\begin{aligned} &\frac{\frac{1}{2}\left[\left(|S[i_1, j_1)| + |S[i_2, j_2)|\right)\left(1 - \frac{1-3\varepsilon}{1-\varepsilon}\right)\right]}{p+q} \\ = &\frac{(p+q)N\left(1 - \frac{1-3\varepsilon}{1-\varepsilon}\right)}{2(p+q)} \\ > &\frac{1}{2}\left(1 - \frac{1-3\varepsilon}{1-\varepsilon}\right) \cdot N. \end{aligned}$$

Notice that this is also a lower bound on the longest common subsequence of B_1 and B_2. This means that

$$\mathrm{ED}(B_1, B_2) < 2N - \left(1 - \frac{1-3\varepsilon/4}{1-\varepsilon/4}\right)N < \frac{2-4\varepsilon}{1-\varepsilon}N < 2(1-\varepsilon)N.$$

This contradicts the error-correcting code's distance property, which we assumed to be larger than $2(1-\varepsilon)N$, and therefore we may conclude that for all indices $i_1 < j_1 \leq i_2 < j_2$ where $j_1 + j_2 - i_1 - i_2 \geq \frac{4N}{\varepsilon}$, (3) holds. □

We point out that even a brute force enumeration of a good insdel code could be used to find a string that satisfies ε-synchronization property for pairs of intervals with large total length. All needed to get an ε-synchronization string is to concatenate that with a string which satisfies ε-synchronization property for small intervals. This one could be brute forced as well. Overall, this gives an alternative polynomial time construction (still using the inspiration of long-distance strings, though). More importantly, if we use a linear time construction for short intervals and a linear time encodable insdel code for long ones, we get a simple $O_\varepsilon(\log n)$-explicit long-distance ε-synchronization string construction for which any interval $[i, i + O_\varepsilon(\log n)]$ is computable in $O_\varepsilon(\log n)$.

In the rest of this section, as depicted in Figure 1, we first introduce a high distance, small alphabet error correcting code that is encodable in linear time in Lemma 4.13 using a high-distance linear-time code introduced in [15]. We then turn this code into a high distance insertion deletion code using the indexing technique from [20]. Finally, we will employ this insertion-deletion code in the setup of Lemma 4.10 to obtain a highly-explicit linear-time long-distance synchronization strings.

Our codes are based on the following code from Guruswami and Indyk [15].

Theorem 4.11 (Theorem 3 from [15]). *For every r, $0 < r < 1$, and all sufficiently small $\epsilon > 0$, there exists a family of codes of rate r and relative distance at least $(1 - r - \epsilon)$ over an alphabet of size $2^{O(\epsilon^{-4} r^{-1} \log(1/\epsilon))}$ such that codes from the family can be encoded in linear time and can also be (uniquely) decoded in linear time from $(1 - r - \epsilon)$ fraction of half-errors, i.e., a fraction e of errors and s of erasures provided $2e + s \leq (1 - r - \epsilon)$.*

One major downside of constructing ε-synchronization strings based on the code from Theorem 4.11 is the exponentially large alphabet size in terms of ε. We concatenate this code with an appropriate small alphabet code to obtain a high-distance code over a smaller alphabet size.

Lemma 4.12. *For sufficiently small ε and $A, R > 1$, and any set Σ_i of size $|\Sigma_i| = 2^{O(\varepsilon^{-5} \log(1/\varepsilon))}$, there exists a code $C : \Sigma_i \longrightarrow \Sigma_o^N$ with distance $1 - \varepsilon$ and rate ε^R where $|\Sigma_o| = O(\varepsilon^{-A})$.*

Proof. To prove the existence of such code, we show that a random code with distance $\delta = 1 - \varepsilon$, rate $r = \varepsilon^A$, alphabet size $|\Sigma_o| = \varepsilon^{-A}$, and block length

$$N = \frac{\log|\Sigma_i|}{\log|\Sigma_o|} \cdot \frac{1}{r} = O\left(\frac{\varepsilon^{-5}\log(1/\varepsilon)}{A\log(1/\varepsilon)} \cdot \frac{1}{\varepsilon^R}\right) = \frac{1}{A} \cdot O\left(\varepsilon^{-5-R}\right)$$

exists with non-zero probability. The probability of two randomly selected codewords of length N out of Σ_o being closer than $\delta = 1 - \varepsilon$ can be bounded above by the following term.

$$\binom{N}{N\varepsilon}\left(\frac{1}{|\Sigma_o|}\right)^{-N\varepsilon}$$

Hence, the probability of the random code with $|\Sigma_o|^{Nr} = |\Sigma_1|$ codewords having a minimum distance smaller than $\delta = 1 - \varepsilon$ is at most the following.

$$\begin{aligned} &\binom{N}{N\varepsilon}\left(\frac{1}{|\Sigma_o|}\right)^{N\varepsilon}\binom{|\Sigma_i|}{2} \\ \leq &\left(\frac{Ne}{N\varepsilon}\right)^{N\varepsilon}\frac{|\Sigma_i|^2}{|\Sigma_o|^{N\varepsilon}} \\ = &\left(\frac{e}{\varepsilon}\right)^{N\varepsilon}\frac{2^{O(\varepsilon^{-5}\log(1/\varepsilon))}}{(\varepsilon^{-A})^{N\varepsilon}} \\ = &2^{O((1-A)\log(1/\varepsilon)N\varepsilon + \varepsilon^{-5}\log(1/\varepsilon))} \\ = &2^{(1-A)O(\varepsilon^{-4-R}\log(1/\varepsilon)) + O(\varepsilon^{-5}\log(1/\varepsilon))} \end{aligned}$$

For $A > 1$, $1 - A$ is negative and for $R > 1$, $\varepsilon^{-4-R}\log(1/\varepsilon)$ is asymptotically larger than $\varepsilon^{-5}\log(1/\varepsilon)$. Therefore, for sufficiently small ε, the exponent is negative and the desired code exists. □

Concatenating the code from Theorem 4.11 (as the outer code) and the code from Lemma 4.12 (as inner code) gives the following code.

Lemma 4.13. *For sufficiently small ε and any constant $0 < \gamma$, there exists an error correcting code of rate $O(\varepsilon^{2.01})$ and distance $1-\varepsilon$ over an alphabet of size $O(\varepsilon^{-(1+\gamma)})$ which is encodable in linear time and also uniquely decodable from an e fraction of erasures and s fraction of symbol substitutions when $s + 2e < 1 - \varepsilon$ in linear time.*

Proof. To construct such code, we simply concatenate codes from Theorem 4.11 and Lemma 4.12 as outer and inner code respectively. Let C_1 be an instantiation of the code from Theorem 4.11 with parameters $r = \varepsilon/4$ and $\epsilon = \varepsilon/4$. Code C_1 is a code of rate $r_1 = \varepsilon/4$ and distance $\delta_1 = 1 - \varepsilon/4 - \varepsilon/4 = 1 - \varepsilon/2$ over an alphabet Σ_1 of size $2^{O(\varepsilon^{-4}r^{-1}\log(1/\varepsilon))} = 2^{O(\varepsilon^{-5}\log(1/\varepsilon))}$ which is encodable and decodable in linear time.

Further, according to Lemma 4.12, one can find a code $C_2 : \Sigma_1 \to \Sigma_2^{N_2}$ for $\Sigma_2 = \varepsilon^{-(1+\gamma)}$ with distance $\delta_2 = 1 - \varepsilon/2$ rate $r_2 = O(\varepsilon^{1.01})$ by performing a brute-force search. Note that block length and alphabet size of C_2 is constant in terms of n. Therefore, such code can be found in $O_\varepsilon(1)$ and by forming a look-up table can be encoded and decoded from δ half-errors in $O(1)$. Hence, concatenating codes C_1 and C_2 gives a code of distance $\delta = \delta_1 \cdot \delta_2 = (1-\varepsilon/2)^2 \geq 1 - \varepsilon$ and rate $r = r_1 \cdot r_2 = O(\varepsilon^{2.01})$ over an alphabet of size $|\Sigma_2| = O\left(\varepsilon^{-(1+\gamma)}\right)$ which can be encoded in linear time in terms of block length and decoded from e fraction of erasures and s fraction of symbol substitutions when $s + 2e < 1 - \varepsilon$ in linear time as well. □

Indexing the codewords of a code from Lemma 4.13 with linear-time constructible synchronization strings of Theorem 4.7 using the technique from [20] summarized in Theorem 3.6 gives Theorem 4.14.

Theorem 4.14. *For sufficiently small ε, there exists a family of insertion-deletion codes with rate $\varepsilon^{O(1)}$ that correct from $1-\varepsilon$ fraction of insertions and deletions over an alphabet of size $\varepsilon^{O(1)}$ that is encodable in linear time and decodable in quadratic time in terms of the block length.*

Proof. Theorem 3.6 provides a technique to convert an error correcting code into an insertion-deletion code by indexing the codewords with a synchronization string. We use the error correcting code C from Lemma 4.13 with parameter $\varepsilon' = \varepsilon/2$ and $\gamma = 0.01$ along with a linear-time constructible synchronization strings S from Theorem 4.7 with parameter $\varepsilon'' = (\varepsilon/2)^2$ in the context of Theorem 3.6. We also use the global decoding algorithm from Haeupler and Shahrasbi [20] for the synchronization string. This will give an insertion deletion code over an alphabet of size $\varepsilon^{O(1)}$ corrects from $(1-\varepsilon') - \sqrt{\varepsilon''} = 1 - \varepsilon$ insdels with a rate of

$$\frac{r_C}{1 + |\Sigma_S|/|\Sigma_C|} = \frac{O\left(\varepsilon^{2.01}\right)}{1 + O(\varepsilon''^{-O(1)}/\varepsilon^{-1.01})} = \varepsilon^{O(1)}.$$

As C is encodable and S is constructible in linear time, the encoding time for the insdel code will be linear. Further, as C is decodable in linear time and S is decodable in quadratic time (using global decoding from [20]), the code is decodable in quadratic time. □

Using insertion-deletion code from Theorem 4.14 and boosting step from Lemma 4.10, we can now proceed to the main theorem of this section that provides a highly explicit construction for $c = O_\varepsilon(1)$-long-distance synchronization strings.

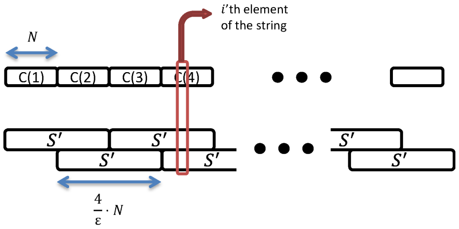

Figure 2: Pictorial representation of the construction of a long-distance ε-synchronization string of length n.

Theorem 4.15. *There is a deterministic algorithm that, for any constant $0 < \varepsilon < 1$ and $n \in \mathbb{N}$, computes a $c = \varepsilon^{-O(1)}$-long-distance ε-synchronization string $S \in \Sigma^n$ where $|\Sigma| = \varepsilon^{-O(1)}$. Moreover, this construction is $O(\log n)$-explicit and can even compute $S[i, i + \log n]$ in $O_\varepsilon(\log n)$ time.*

Proof. We simply use an insertion-deletion code from Theorem 4.14 with parameter $\varepsilon' = \varepsilon/4$ and block length $N = \frac{\log_q n}{R}$ where $q = \varepsilon^{-O(1)}$ is the size of the alphabet from Theorem 4.14. Using this code in Lemma 4.10 gives a string S of length $q^{RN} \cdot N \geq n$ that satisfies $4\varepsilon' = \varepsilon$-synchronization property over any pair of intervals of total length $\frac{4N}{\varepsilon} = O\left(\frac{\log n}{\varepsilon R \log q}\right) = O\left(\varepsilon^{-O(1)} \log n\right)$ or more. Since the insertion-deletion code from Theorem 4.14 is linearly encodable, the construction will be highly-explicit.

To turn S into a c-long-distance ε-synchronization string for $c = \frac{4N}{\varepsilon \log n} = O\left(\varepsilon^{-O(1)}\right)$, we simply concatenate it with a string T that satisfies ε-synchronization property for neighboring intervals of total size smaller than $c \log n$. In other words, we propose the following structure for constructing c-long-distance ε-synchronization string R.

$$R[i] = (S[i], T[i]) = \left(C\left(\left\lfloor \frac{i}{N} \right\rfloor\right)[i \,(\mathrm{mod}\, N)], T[i]\right) \tag{4}$$

Let S' be an ε-synchronization string of length $2c \log n$. Using linear-time construction from Theorem 4.7, one can find S' in linear time in its length, i.e, $O(\log n)$. We define strings T_1 and T_2 consisting of repetitions of S' as follows.

$$T_1 = (S', S', \cdots, S'), \qquad T_2 = (0^{c \log n}, S', S', \cdots, S')$$

The string $T_1 \cdot T_2$ satisfies ε-synchronization strings for neighboring intervals of total length $c \log n$ or less as any such substring falls into one copy of S'. Note that having S' one can find any symbol of T in linear time. Hence, T has a highly-explicit linear time construction. Therefore, concatenating S and T gives a linear time construction for c-long-distance ε-synchronization strings over an alphabet of size $\varepsilon^{-O(1)}$ that is highly-explicit and, further, allows computing any substring $[i, i + \log n]$ in $O(\log n)$ time. A schematic representation of this construction can be found in Figure 2. □

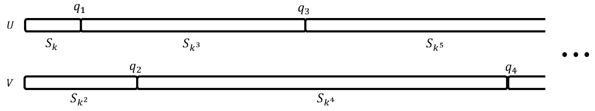

Figure 3: Construction of Infinite synchronization string T

4.5 Infinite Synchronization Strings: Highly Explicit Construction

Throughout this section we focus on construction of infinite synchronization strings. To measure the efficiency of a an infinite string's construction, we consider the required time complexity for computing the first n elements of that string. Moreover, besides the time complexity, we employ a generalized notion of explicitness to measure the quality of infinite string constructions.

In a similar fashion to finite strings, an infinite synchronization string is called to have a $T(n)$-*explicit* construction if there is an algorithm that computes any position $S[i]$ in $O(T(i))$. Moreover, it is said to have a highly-explicit construction if $T(i) = O(\log i)$.

We show how to deterministically construct an infinitely-long ε-synchronization string over an alphabet Σ which is polynomially large in ε^{-1}. Our construction can compute the first n elements of the infinite string in $O(n)$ time, is highly-explicit, and, further, can compute any $[i, i + \log i]$ in $O(\log i)$.

Theorem 4.16. *For all $0 < \varepsilon < 1$, there exists an infinite ε-synchronization string construction over a poly(ε^{-1})-sized alphabet that is highly-explicit and also is able to compute $S[i, i + \log i]$ in $O(\log i)$. Consequently, using this construction, the first n symbols of the string can be computed in $O(n)$ time.*

Proof. Let $k = \frac{6}{\varepsilon}$ and let S_i denote a $\frac{\varepsilon}{2}$-synchronization string of length i. We define U and V as follows:

$$U = (S_k, S_{k^3}, S_{k^5}, \dots), \qquad V = (S_{k^2}, S_{k^4}, S_{k^6}, \dots)$$

In other words, U is the concatenation of $\frac{\varepsilon}{2}$-synchronization strings of length $k, k^3, k^5, \dots$ and V is the concatenation of $\frac{\varepsilon}{2}$-synchronization strings of length $k^2, k^4, k^6, \dots$. We build an infinite string T such that $T[i] = (U[i], V[i])$ (see Figure 3).

First, if finite synchronization strings S_{k^l} used above are constructed using the highly-explicit construction algorithm introduced in Theorem 4.15, any index i can be computed by simply finding one index in two of S_{k^l}s in $O(\log n)$. Further, any substring of length n of this construction can be computed by constructing finite synchronization strings of total length $O(n)$. According to Theorem 4.15, that can be done in $O_\varepsilon(n)$.

Now, all that remains is to show that T is an ε-synchronization string. We use following lemma to prove this.

Lemma 4.17. *Let $x < y < z$ be positive integers and let t be such that $k^t \le |T[x, z]| < k^{t+1}$. Then there exists a block of S_{k^i} in U or V such that all but a $\frac{3}{k}$ fraction of $T[x, z)$ is covered by S_{k^i}.*

Note that this lemma shows that

$$\begin{aligned} ED(T[x, y), T[y, z)) &> (1 - \varepsilon/2)\,(|T[x, y)| + |T[y, z)|)\,(1 - 3/k) \\ &= (1 - \varepsilon/2)^2\,(|T[x, y)| + |T[y, z)|) \\ &\ge (1 - \varepsilon)\,(|T[x, y)| + |T[y, z)|) \end{aligned}$$

which implies that T is an ε-synchronization string. □

Proof of Lemma 4.17. We first define i^{th} *turning point* q_i to be the index of T at which $S_{k^{i+1}}$ starts, i.e., $q_i = k^i + k^{i-2} + k^{i-4} + \cdots$. Note that

$$q_i = \begin{cases} k^2 + k^4 + \cdots + k^i & \text{Even } i \\ k + k^3 + \cdots + k^i & \text{Odd } i \end{cases} \tag{5}$$

$$= \begin{cases} k^2 \frac{k^i - 1}{k^2 - 1} & \text{Even } i \\ k \frac{k^{i+1} - 1}{k^2 - 1} & \text{Odd } i \end{cases} \tag{6}$$

Note that $q_{t-1} < 2k^{t-1}$ and $|T[x, z)| \ge k^t$. Therefore, one can throw away all the elements of $T[x, z)$ whose indices are less than q_{t-1} without losing more than a $\frac{2}{k}$ fraction of the elements of $T[x, z)$. We will refer to the remaining part of $T[x, z)$ as $\tilde{T}$.

Now, the distance of any two turning points q_i and q_j where $t \le i < j$ is at least $q_{t+1} - q_t$, and

$$q_{t+1} - q_t = \begin{cases} k \frac{k^{t+2} - 1}{k^2 - 1} - k^2 \frac{k^t - 1}{k^2 - 1} & \text{Even } t \\ k^2 \frac{k^{t+1} - 1}{k^2 - 1} - k \frac{k^{t+1} - 1}{k^2 - 1} & \text{Odd } t \end{cases} \tag{7}$$

$$= \begin{cases} \frac{(k-1)(k^{t+2} + k)}{k^2 - 1} = \frac{k^{t+2} + k}{k+1} & \text{Even } t \\ \frac{(k-1)(k^{t+2} - k)}{k^2 - 1} = \frac{k^{t+2} - k}{k+1} & \text{Odd } t. \end{cases} \tag{8}$$

Hence, $q_{t+1} - q_t > k^{t+1}\left(1 - \frac{1}{k}\right)$. Since $|\tilde{T}| \le |T[x, z)| < k^{t+1}$, this fact gives that there exists a S_{k^i} which covers a $\left(1 - \frac{1}{k}\right)$ fraction of $\tilde{T}$. This completes the proof of the lemma. □

5 LOCAL DECODING

In Section 4, we discussed the close relationship between long-distance synchronization strings and insertion-deletion codes and provided highly-explicit constructions of long-distance synchronization strings based on insdel codes.

In this section, we make a slight modification to the highly explicit structure (4) we introduced in Theorem 4.15 where we showed one can use a constant rate insertion-deletion code C with distance $1 - \frac{\varepsilon}{4}$ and block length $N = O(\log n)$ and a string T satisfying ε-synchronization property for pairs of neighboring intervals of total length $c \log n$ or less to make a c-long-distance synchronization string of length n. In addition to the symbols of the string consisting of codewords of C and symbols of string T, we append $\Theta\left(\log \frac{1}{\varepsilon}\right)$ extra bits to each symbol to enable *local decodability*. This extra symbol, as described in (9), essentially works as a circular index counter for insertion-deletion code blocks.

$$R[i] = \left(C\left(\left\lfloor \frac{i}{N} \right\rfloor\right)[i\,(\text{mod } N)]\,, T[i], \left\lfloor \frac{i}{N} \right\rfloor \left(\text{mod } \frac{8}{\varepsilon^3}\right)\right) \tag{9}$$

With this extra information appended to the construction, we claim that *relative suffix error density* is smaller than ε upon arrival of some symbol, then one can decode the corresponding index correctly by only looking at the last $O(\log n)$ symbols. At any point of a communication over an insertion-deletion channel, relative suffix error density is defined as the maximum fraction of errors occurred over all suffixes of the message sent so far. (Definition 5.12 from [20]).

THEOREM 5.1. *Let R be a highly-explicit long-distance ε-synchronization string constructed according to (9). Let $R[1,i]$ be sent by Alice and be received as $R'[1,j]$ by Bob. If relative suffix error density is smaller than $1-\frac{\varepsilon}{2}$, then Bob can find i in $\frac{4}{\varepsilon}\cdot T_{Dec}(N)+\frac{4N}{\varepsilon}\cdot(T_{Enc}(N)+Ex_T(c\log n)+c^2\log^2 n)$ only by looking at the last $\max(\frac{4N}{\varepsilon^2}, c\log n)$ received symbols where T_{Enc} and T_{Dec} is the encoding and decoding complexities of C and $Ex_T(l)$ is the amount of time it takes to construct a substring of T of length l.*

For linear-time encodable, quadratic-time decodable code C and highly-explicit string T constructed by repetitions of short synchronization strings used in Theorem 4.15, construction (9) provides the following.

THEOREM 5.2. *Let R be a highly-explicit long-distance ε-synchronization string constructed according to (9) with code C and string T as described in Theorem 4.15. Let $R[1,i]$ be sent by Alice and be received as $R'[1,j]$ by Bob. If relative suffix error density is smaller than $1-\frac{\varepsilon}{2}$, then Bob can find i in $O(\log^3 n)$ time only by looking at the last $O(\log n)$ received symbols.*

This decoding procedure, which we will refer to as *local decoding* consists of two principal phases upon arrival of each symbol. During the first phase, the receiver finds a list of $\frac{1}{\varepsilon}$ numbers that is guaranteed to contain the index of the current insertion-deletion code block. This gives $\frac{N}{\varepsilon}$ candidates for the index of the received symbol. The second phase uses the relative suffix error density guarantee to choose the correct candidate among the list. The following lemma formally presents the first phase. This idea of using list decoding as a middle step to achieve unique decoding has been used by several previous work [12, 16–18].

LEMMA 5.3. *Let S be an ε-synchronization string constructed as described in (9). Let $S[1,i]$ be sent by Alice and be received as $S_\tau[1,j]$ by Bob. If relative suffix error density is smaller than $1-\varepsilon/2$, then Bob can compute a list of $\frac{4N}{\varepsilon}$ numbers that is guaranteed to contain i.*

PROOF. Note that as relative suffix error density is smaller than $1-\varepsilon/2<1$, the last received symbol has to be successfully transmitted. Therefore, Bob can correctly figure out the insertion-deletion code block index counter value which we denote by *count*. Note that if there are no errors, all symbols in blocks with index counter value of $count, count-1,\cdots,count-4/\varepsilon+1 \bmod \frac{8}{\varepsilon^3}$ that was sent by Bob right before the current symbol, have to be arrived within the past $4/\varepsilon\cdot N$ symbols. However, as adversary can insert symbols, those symbols can appear anywhere within the last $\frac{2}{\varepsilon}\frac{4N}{\varepsilon}=\frac{8N}{\varepsilon^2}$ symbols.

Hence, if Bob looks at the symbols arrived with index $i\in\{count, count-1,\cdots,count-4/\varepsilon+1\} \bmod \frac{8}{\varepsilon^3}$ within the last $\frac{8N}{\varepsilon^2}$ received symbols, he can observe all symbols coming from blocks with index $count, count-1,\cdots,count-4/\varepsilon+1 \bmod \frac{8}{\varepsilon^3}$ that was sent right before $S[i]$. Further, as our counter counts modulo $\frac{8}{\varepsilon^3}$, no symbols from older blocks with indices $count, count-1,\cdots,count-1/\varepsilon+1 \bmod \frac{4}{\varepsilon^3}$ will appear within the past $\frac{8N}{\varepsilon^2}$ symbols. Therefore, Bob can find the symbols from the last $\frac{4}{\varepsilon}$ blocks up to some insdel errors. By decoding those blocks, he can make up a list of $\frac{4}{\varepsilon}$ candidates for the actual block number. As each block contains N elements, there are a total of $\frac{4N}{\varepsilon}$ many candidates for i.

Note that as relative suffix error density is at most $1-\varepsilon/2$ and the last block may not have been completely sent yet, the total fraction of insdels in reconstruction of the last $\frac{4}{\varepsilon}$ blocks on Bob side smaller than $1-\varepsilon/2+\frac{N}{4N/\varepsilon^2}\le 1-\frac{\varepsilon}{4}$. Therefore, the error density in at least one of those blocks is not larger than $1-\frac{\varepsilon}{4}$. This guarantees that at least one block will be correctly decoded and henceforth the list contains the correct actual index. □

We now define a limited version of relative suffix distance (defined in [20]) which enables us to find the correct index among candidates found in Lemma 5.3.

Definition 5.4 (Limited Relative Suffix Distance). For any two strings $S,S'\in\Sigma^*$ we define their l-limited relative suffix distance, l-LRSD, as follows:

$$l\text{-LRSD}(S,S')=\max_{0<k<l}\frac{\mathrm{ED}\left(S(|S|-k,|S|],S'(|S'|-k,|S'|]\right)}{2k}$$

Note that $l=O(\log n)$-limited suffix distance of two strings can be computed in $O(l^2)=O(\log^2 n)$ by computing edit distance of all pairs of prefixes of their l-long suffixes.

LEMMA 5.5. *If string S is a c-long distance ε-synchronization string, then for any two distinct prefixes $S[1,i]$ and $S[1,j]$, $(c\log n)$-LRSD$(S[1,i],S[1,j])>1-\varepsilon$.*

PROOF. If $j-i<c\log n$, the synchronization string property gives that $\mathrm{ED}(S(2i-j,i],S(i,j])>2(j-i)(1-\varepsilon)$ which gives the claim for $k=j-i$. If $j-i\ge c\log n$, the long-distance property gives that $\mathrm{ED}(S(i-\log n,i],S(j-\log n,j])>2(1-\varepsilon)c\log n$ which again, proves the claim. □

Lemmas 5.3 and 5.5 enable us to prove Theorem 5.1.

PROOF OF THEOREM 5.1. Using Lemma 5.3, by decoding $4/\varepsilon$ codewords, Bob forms a list of $4N/\varepsilon$ candidates for the index of the received symbol. This will take $4/\varepsilon\cdot T_{Dec}(N)$ time. Then, using Lemma 5.5, for any of the $4N/\varepsilon$ candidates, he has to construct a $c\log n$ substring of R and compute the $(c\log n)$-LRSD of that with the string he received. This requires looking at the last $\max(4n/\varepsilon, c\log n)$ recieved symbols and takes $4N/\varepsilon\cdot(T_{Enc}(N)+Ex_T(c\log n)+c^2\log^2 n)$ time. □

6 APPLICATION: NEAR LINEAR TIME CODES AGAINST INSDELS, BLOCK TRANSPOSITIONS, AND BLOCK REPLICATIONS

In Sections 4 and 5, we provided highly explicit constructions and local decodings for synchronization strings. Utilizing these two important properties of synchronization strings together suggests important improvements over insertion-deletion codes introduced by Haeupler and Shahrasbi [20]. We start by stating the following important lemma which summarizes the results of Sections 4 and 5.

LEMMA 6.1. *For any $0<\varepsilon<1$, there exists an streaming (n,δ)-indexing solution with ε-synchronization string S and streaming decoding algorithm $\mathcal{D}$ that figures out the index of each symbol by merely considering the last $O_\varepsilon(\log n)$ received symbols and in $O_\varepsilon(\log^3 n)$*

time. Further, $S \in \Sigma^n$ is highly-explicit and constructible in linear-time and $|\Sigma| = O\left(\varepsilon^{-O(1)}\right)$. This solution may contain up to $\frac{n\delta}{1-\varepsilon}$ misdecodings.

Proof. Let S be a long-distance 2ε-synchronization string constructed according to Theorem 4.15 and enhanced as suggested in (9) to ensure local decodablity. As discussed in Sections 4 and 5, these strings trivially satisfy all properties claimed in the statement other than the misdecoding guarantee.

According to Theorem 5.2, correct decoding is ensured whenever relative suffix error density is less than $1 - \frac{2\varepsilon}{2} = 1 - \varepsilon$. Therefore, as relative suffix error density can exceed $1 - \varepsilon$ upon arrival of at most $\frac{n\delta}{1-\varepsilon}$ many symbols (see Lemma 5.14 from [20]), there can be at most $\frac{n\delta}{1-\varepsilon}$ many successfully received symbols which are not decoded correctly. This proves the misdecoding guarantee. □

6.1 Near-Linear Time Insertion-Deletion Code

Using the indexing technique proposed by Haeupler and Shahrasbi [20] summarized in Theorem 3.6 with synchronization strings and decoding algorithm from Theorem 3.5, one can obtain the following insdel codes.

Theorem 6.2. *For any $0 < \delta < 1/3$ and sufficiently small $\varepsilon > 0$, there exists an encoding map $E : \Sigma^k \to \Sigma^n$ and a decoding map $D : \Sigma^* \to \Sigma^k$, such that, if $EditDistance(E(m), x) \le \delta n$ then $D(x) = m$. Further, $\frac{k}{n} > 1 - 3\delta - \varepsilon$, $|\Sigma| = f(\varepsilon)$, and E and D can be computed in $O(n)$ and $O(n \log^3 n)$ time respectively.*

Proof. We closely follow the proof of Theorem 1.1 from [20] and use Theorem 3.6 to convert a near-MDS error correcting code to an insertion-deletion code satisfying the claimed properties.

Given the δ and ε, we choose $\varepsilon' = \frac{\varepsilon}{12}$ and use locally decodable $O_{\varepsilon'}(1)$-long-distance ε'-synchronization string S of length n over alphabet Σ_S of size $\varepsilon'^{-O(1)} = \varepsilon^{-O(1)}$ from Theorem 5.2.

We plug this synchronization string with the local decoding from Theorem 5.2 into Theorem 3.6 with a near-MDS expander code [15] C (see Theorem 4.11) which can efficiently correct up to $\delta_C = 3\delta + \frac{\varepsilon}{3}$ half-errors and has a rate of $R_C > 1 - \delta_C - \frac{\varepsilon}{3}$ over an alphabet $\Sigma_C = \exp(\varepsilon^{-O(1)})$ such that $\log|\Sigma_C| \ge \frac{3\log|\Sigma_S|}{\varepsilon}$. This ensures that the final rate is indeed at least $\frac{R_C}{1+\frac{\log\Sigma_S}{\log\Sigma_C}} \ge R_C - \frac{\log\Sigma_S}{\log\Sigma_C} = 1 - 3\delta - 3\frac{\varepsilon}{3} = 1 - 3\delta - \varepsilon$ and the fraction of insdel errors that can be efficiently corrected is $\delta_C - 2\frac{\delta}{1-\varepsilon'} \ge 3\delta + \varepsilon/3 - 2\delta(1 + 2\varepsilon') \ge \delta$. The encoding and decoding complexities are furthermore straight forward according to guarantees stated in Theorem 6.1 and the linear time construction of S. □

6.2 Insdels, Block Transpositions, and Block Replications

In this section, we introduce block transposition and block replication errors and show that code from Theorem 6.2 can overcome these types errors as well.

One can think of several way to model transpositions and replications of blocks of data. One possible model would be to have the string of data split into blocks of length l and then define transpositions and replications over those fixed blocks. In other words, for message $m_1, m_2, \cdots, m_n \in \Sigma^n$, a single transposition or replication would be defined as picking a block of length l and then move or copy that blocks of data somewhere in the message.

Another (more general) model is to let adversary choose any block, i.e., substring of the message he wishes and then move or copy that block somewhere in the string. Note that in this model, a constant fraction of block replications may make the message length exponentially large in terms of initial message length. We will focus on this more general model and provide codes protecting against them running near-linear time in terms of the received block length. Such results automatically extend to the weaker model that does not lead to exponentially large corrupted messages.

We now formally define (i, j, l)-*block transposition* as follows.

Definition 6.3 ((i, j, l)-Block Transposition). For a given string $M = m_1 \cdots m_n$, the (i, j, l)-*block transposition* operation for $1 \le i \le i + l \le n$ and $j \in \{1, \cdots, i - 1, i + l + 1, \cdots, n\}$ is defined as an operation which turns M into

$$M' = m_1, \cdots, m_{i-1}, m_{i+l+1} \cdots, m_j, m_i \cdots m_{i+l}, m_{j+1}, \cdots, m_n$$

if $j > i + l$ or

$$M' = m_1, \cdots, m_j, m_i, \cdots, m_{i+l}, m_{j+1}, \cdots, m_{i-1}, m_{i+l+1} \cdots, m_n$$

if $j < i$ by removing $M[i, i + l]$ and inserting it right after $M[j]$.

Also, (i, j, l)-*block replication* is defined as follows.

Definition 6.4 ((i, j, l)-Block Replication). For a given string $M = m_1 \cdots m_n$, the (i, j, l)-*block replication* operation for $1 \le i \le i+l \le n$ and $j \in \{1, \cdots, n\}$ is defined as an operation which turns M into $M' = m_1, \cdots, m_j, m_i \cdots m_{i+l}, m_{j+1}, \cdots, m_n$ which is obtained by copying $M[i, i + l]$ right after $M[j]$.

We now proceed to the following theorem that implies the code from Theorem 6.2 recovers from block transpositions and replications as well.

Theorem 6.5. *Let $S \in \Sigma_S^n$ be a locally-decodable highly-explicit c-long-distance ε-synchronization string from Theorem 5.2 and C be an half-error correcting code of block length n, alphabet Σ_C, rate r, and distance d with encoding function $\mathcal{E}_C$ and decoding function $\mathcal{D}_C$ that run in $T_{\mathcal{E}_C}$ and $T_{\mathcal{D}_C}$ respectively. Then, one can obtain an encoding function $E_n : \Sigma_C^{nr} \to [\Sigma_C \times \Sigma_S]^n$ that runs in $T_{\mathcal{E}_C} + O(n)$ and decoding function $D_n : [\Sigma_C \times \Sigma_S]^* \to \Sigma_C^{nr}$ which runs in $T_{\mathcal{D}_C} + O\left(\log^3 n\right)$ and recovers from $n\delta_{insdel}$ fraction of synchronization errors and δ_{block} fraction of block transpositions or replications as long as $\left(2 + \frac{2}{1-\varepsilon/2}\right)\delta_{insdel} + (12c\log n)\delta_{block} < d$.*

Proof. To obtain such codes, we simply index the symbols of the given error correcting code with the symbols of the given synchronization strings. More formally, the encoding function $\mathcal{E}(x)$ for $x \in \Sigma_C^{nr}$ first computes $\mathcal{E}_C(x)$ and then indexes it, symbol by symbol, with the elements of the given synchronization string.

On the decoding end, $\mathcal{D}(x)$ first uses the indices on each symbol to guess the actual position of the symbols using the local decoding of the c-long-distance ε-synchronization string. Rearranging the received symbols in accordance to the guessed indices, the receiving end obtains a version of $\mathcal{E}_C(x)$, denoted by $\bar{x}$, that may suffer from a number of symbol corruption errors due to incorrect index misdecodings. As long as the number of such misdecodings, k, satisfies $n\delta_{insdel} + 2k \le nd$, computing $\mathcal{D}_C(\bar{x})$ gives x. The decoding procedure naturally consists of decoding the attached synchronization string, rearranging the indices, and running $\mathcal{D}_C$ on the rearranged version. Note that if multiple symbols where detected to be located at the same position by the synchronization string decoding procedure or no symbols where detected to be at some position, the decoder can simply put a special symbol '?' there and treat it as a half-error. The decoding and encoding complexities are trivial.

In order to find the actual index of a received symbol correctly, we need the local decoding procedure to compute the index correctly. For that purpose, it suffices that no block operations cut or paste symbols within an interval of length $2c\log n$ before that index throughout the entire block transpositions/replications performed by the adversary and the relative suffix error density caused by synchronization errors for that symbol does not exceed $1-\varepsilon/2$. As any block operation might cause three new cut/cop/paste edges and relative suffix error density is larger than $1-\varepsilon/2$ for up to $\frac{1}{1-\varepsilon/2}$ many symbols (according to Lemma 5.14 from [20]), the positions of all but at most $k \le 3n\delta_{block} \times 2c\log n + n\delta_{insdel}\left(1+\frac{1}{1-2\varepsilon}\right)$ symbols will be decoded incorrectly via synchronization string decoding procedure. Hence, as long as $n\delta_{insdel} + 2k \le 6\delta_{block} \times 2c\log n + n\delta_{insdel}\left(3+\frac{2}{1-2\varepsilon}\right) < d$ the decoding procedure succeeds. Finally, the encoding and decoding complexities follow from the fact that indexing codewords of length n takes linear time and the local decoding of synchronization strings takes $O(n\log^3 n)$ time. □

Employing locally-decodable $O_\varepsilon(1)$-long-distance synchronization strings of Theorem 5.2 and error correcting code of Theorem 4.11 in Theorem 6.5 gives the following code.

Theorem 6.6. *For any $0 < r < 1$ and sufficiently small ε there exists a code with rate r that corrects $n\delta_{insdel}$ synchronization errors and $n\delta_{block}$ block transpositions or replications as long as $6\delta_{insdel} + c\log n\delta_{block} < 1-r-\varepsilon$ for some $c = O(1)$. The code is over an alphabet of size $O_\varepsilon(1)$ and has $O(n)$ encoding and $O(N\log^3 n)$ decoding complexities where N is the length of the received message.*

7 APPLICATIONS: NEAR-LINEAR TIME INFINITE CHANNEL SIMULATIONS WITH OPTIMAL MEMORY CONSUMPTION

We now show that the indexing algorithm introduced in Theorem 6.1 can improve the efficiency of channel simulations from [23] as well as insdel codes. Consider a scenario where two parties are maintaining a communication that suffers from synchronization errors, i.e, insertions and deletions. Haeupler et al. [23] provided a simple technique to overcome this desynchronization. Their solution consists of a simple symbol by symbol attachment of a synchronization string to any transmitted symbol. The attached indices enables the receiver to correctly detect indices of most of the symbols he receives. However, the decoding procedure introduced in Haeupler et al. [23] takes polynomial time in terms of the communication length. The explicit construction introduced in Section 4 and local decoding provided in Section 5 can reduce the construction and decoding time and space complexities to polylogarithmic. Further, the decoding procedure only requires to look up $O_\varepsilon(\log n)$ recently received symbols upon arrival of any symbol.

Interestingly, we will show that, beyond the time and space complexity improvements over simulations in [23], long-distance synchronization strings can make *infinite channel simulations* possible. In other words, two parties communicating over an insertion-deletion channel are able to simulate a corruption channel on top of the given channel even if they are not aware of the length of the communication before it ends with similar guarantees as of [23]. To this end, we introduce infinite strings that can be used to index communications to convert synchronization errors into symbol corruptions. The following theorem analogous to the indexing algorithm of Lemma 6.1 provides all we need to perform such simulations.

Theorem 7.1. *For any $0 < \varepsilon < 1$, there exists an infinite string S that satisfies the following properties:*

1. *String S is over an alphabet of size $\varepsilon^{-O(1)}$.*
2. *String S has a highly-explicit construction and, for any i, $S[i, i+\log i]$ can be computed in $O(\log i)$.*
3. *Assume that $S[1,i]$ is sent over an insertion-deletion channel. There exists a decoding algorithm for the receiving side that, if relative suffix error density is smaller than $1-\varepsilon$, can correctly find i by looking at the last $O(\log i)$ and knowing the number of received symbols in $O(\log^3 i)$ time.*

Proof. To construct such a string S, we use our finite-length highly-explicit locally-decodable long-distance synchronization string constructions from Theorem 5.2 and use to construct finite substrings of S as proposed in the infinite string construction of Theorem 4.16 which is depicted in Figure 3. We choose length progression parameter $k = 10/\varepsilon^2$. Similar to the proof of Lemma 4.17, we define *turning point* q_i as the index at which $S_{k^{i+1}}$ starts. We append one extra bit to each symbol $S[i]$ which is zero if $q_j \le i < q_{j+1}$ for some even j and one otherwise.

This construction clearly satisfies the first two properties claimed in the theorem statement. To prove the third property, suppose that $S[1,i]$ is sent and received as $S'[1,i']$ and the error suffix density is less than $1-\varepsilon$. As error suffix density is smaller than $1-\varepsilon$, $i\varepsilon \le i' \le i/\varepsilon$ which implies that $i'\varepsilon \le i \le i'/\varepsilon$. This gives an uncertainty interval whose ends are close by a factor of $1/\varepsilon^2$. By the choice of k, this interval contains at most one turning point. Therefore, using the extra appended bit, receiver can figure out index j for which $q_j \le i < q_{j+1}$. Knowing this, it can simply use the local decoding algorithm for finite string S_{j-1} to find i. □

Theorem 7.2. *[Improving Channel Simulations from [23]]*

(a) *Suppose that n rounds of a one-way/interactive insdel channel over an alphabet Σ with a δ fraction of insertions and deletions are given. Using an ε-synchronization string over alphabet Σ_{syn}, it is possible to simulate $n(1-O_\varepsilon(\delta))$ rounds of a one-way/interactive corruption channel over Σ_{sim} with at most $O_\varepsilon(n\delta)$ symbols corrupted so long as $|\Sigma_{sim}| \times |\Sigma_{syn}| \le |\Sigma|$.*

(b) *Suppose that n rounds of a binary one-way/interactive insertion-deletion channel with a δ fraction of insertions and deletions are given. It is possible to simulate $n(1-\Theta(\sqrt{\delta\log(1/\delta)}))$ rounds of a binary one-way/interactive corruption channel* *with $\Theta(\sqrt{\delta \log(1/\delta)})$ fraction of corruption errors between two parties over the given channel.*

Having an explicitly-constructible, locally-decodable, infinite string from Theorem 7.1 utilized in the simulation, all of the simulations mentioned above take $O(\log n)$ time for sending/starting party of one-way/interactive communications. Further, on the other side, the simulation spends $O(\log^3 n)$ time upon arrival of each symbol and only looks up $O(\log n)$ many recently received symbols. Overall, these simulations take a $O(n \log^3 n)$ time and $O(\log n)$ space to run. These simulations can be performed even if parties are not aware of the communication length.

Proof. We simply replace ordinary ε-synchronization strings used in all such simulations in [23] with the highly-explicit locally-decodable infinite string from Theorem 7.1 with its corresponding local-decoding procedure instead of minimum RSD decoding procedure that is used in [23]. This keeps all properties that simulations proposed by Haeupler et. al. [23] guarantee. Further, by properties stated in Theorem 7.1, the simulation is performed in near-linear time, i.e., $O(n \log^3 n)$. Also, constructing and decoding each symbol of the string from Theorem 7.1 only takes $O(\log n)$ space which leads to an $O(\log n)$ memory requirement on both sides. □

8 APPLICATIONS: NEAR-LINEAR TIME CODING SCHEME FOR INTERACTIVE COMMUNICATION

Using the near-linear time interactive channel simulation in Theorem 7.2 with the near-linear time interactive coding scheme of Haeupler and Ghaffari [12] (stated in Theorem 8.1) gives the near-linear time coding scheme for interactive communication over insertion-deletion channels stated in Theorem 8.2.

Theorem 8.1 (Theorem 1.1 from [12]). *For any constant $\varepsilon > 0$ and n-round protocol Π there is a randomized non-adaptive coding scheme that robustly simulates Π against an adversarial error rate of $\rho \leq 1/4 - \varepsilon$ using $N = O(n)$ rounds, a near-linear $n \log^{O(1)} n$ computational complexity, and failure probability $2^{-\Theta(n)}$.*

Theorem 8.2. *For a sufficiently small δ and n-round alternating protocol Π, there is a randomized coding scheme simulating Π in presence of δ fraction of edit-corruptions with constant rate (i.e., in $O(n)$ rounds) and in near-linear time. This coding scheme works with probability $1 - 2^{\Theta(n)}$.*

REFERENCES

[1] Noga Alon, Jeff Edmonds, and Michael Luby. 1995. Linear time erasure codes with nearly optimal recovery. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 512–519.

[2] Arturs Backurs and Piotr Indyk. 2015. Edit distance cannot be computed in strongly subquadratic time (unless seth is false). In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*. ACM, 51–58.

[3] J Beck. 1984. An application of Lovász local lemma: there exists an infinite 01-sequence containing no near identical intervals. In *Finite and Infinite Sets*. Elsevier, 103–107.

[4] Zvika Brakerski and Yael Tauman Kalai. 2012. Efficient interactive coding against adversarial noise. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 160–166.

[5] Zvika Brakerski and Moni Naor. 2013. Fast algorithms for interactive coding. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 443–456.

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

[7] Mark Braverman and Anup Rao. 2014. Toward coding for maximum errors in interactive communication. *IEEE Transactions on Information Theory* 60, 11 (2014), 7248–7255.

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

[9] Kuan Cheng, Xin Li, and Ke Wu. 2017. Synchronization Strings: Efficient and Fast Deterministic Constructions over Small Alphabets. *arXiv preprint arXiv:1710.07356* (2017).

[10] Matthew Franklin, Ran Gelles, Rafail Ostrovsky, and Leonard J Schulman. 2015. Optimal coding for streaming authentication and interactive communication. *IEEE Transactions on Information Theory* 61, 1 (2015), 133–145.

[11] Ran Gelles, Ankur Moitra, and Amit Sahai. 2014. Efficient coding for interactive communication. *IEEE Transactions on Information Theory* 60, 3 (2014), 1899–1913.

[12] Mohsen Ghaffari and Bernhard Haeupler. 2014. Optimal error rates for interactive coding II: Efficiency and list decoding. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 394–403.

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

[14] Venkatesan Guruswami and Piotr Indyk. 2001. Expander-based constructions of efficiently decodable codes. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 658–667.

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

[16] Venkatesan Guruswami and Ray Li. 2016. Efficiently decodable insertion/deletion codes for high-noise and high-rate regimes. In *Proceedings of the 2016 IEEE International Symposium on Information Theory*.

[17] Venkatesan Guruswami and Atri Rudra. 2006. Explicit capacity-achieving list-decodable codes. In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*. ACM, 1–10.

[18] Venkatesan Guruswami and Carol Wang. 2015. Deletion Codes in the High-noise and High-rate Regimes. In *Proceedings of the 19th International Workshop on Randomization and Computation (RANDOM)*. 867–880.

[19] Bernhard Haeupler. 2014. Interactive channel capacity revisited. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*. 226–235.

[20] Bernhard Haeupler and Amirbehshad Shahrasbi. 2017. Synchronization strings: codes for insertions and deletions approaching the Singleton bound. In *Proceedings of the Annual Symposium on Theory of Computing (STOC)*. ACM, 33–46.

[21] Bernhard Haeupler and Amirbehshad Shahrasbi. 2017. Synchronization Strings: Explicit Constructions, Local Decoding, and Applications. *arXiv preprint arXiv:1710.09795* (2017).

[22] 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)*.

[23] 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)*.

[24] Brett Hemenway, Noga Ron-Zewi, and Mary Wootters. 2017. Local List Recovery of High-Rate Tensor Codes & Applications. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*.

[25] Gillat Kol and Ran Raz. 2013. Interactive channel capacity. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. ACM, 715–724.

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

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

[28] Leonard J. Schulman. 1992. Communication on noisy channels: A coding theorem for computation. In *Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS)*. 724–733.

[29] Leonard J. Schulman. 1996. Coding for interactive communication. *IEEE transactions on information theory* 42, 6 (1996), 1745–1756.

[30] Leonard J. Schulman and David Zuckerman. 1999. Asymptotically good codes correcting insertions, deletions, and transpositions. *IEEE transactions on information theory* 45, 7 (1999), 2552–2557.

[31] Alexander A Sherstov and Pei Wu. 2017. Optimal Interactive Coding for Insertions, Deletions, and Substitutions.. In *Electronic Colloquium on Computational Complexity (ECCC)*, Vol. 24. 79.

[32] Michael Sipser and Daniel A Spielman. 1996. Expander codes. *IEEE Transactions on Information Theory* 42, 6 (1996), 1710–1722.

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

[34] Daniel Alan Spielman. 1995. *Computationally efficient error-correcting codes and holographic proofs.* Ph.D. Dissertation. Massachusetts Institute of Technology.