Synchronization Strings and Codes for Insertions and Deletions – a Survey

Bernhard Haeupler and Amirbehshad Shahrasbi

***Abstract*—Already in the 1960s, Levenshtein and others studied error-correcting codes that protect against synchronization errors, such as symbol insertions and deletions. However, despite significant efforts, progress on designing such codes has been lagging until recently, particularly compared to the detailed understanding of error-correcting codes for symbol substitution or erasure errors. This paper surveys the recent progress in designing efficient error-correcting codes over finite alphabets that can correct a constant fraction of worst-case insertions and deletions.**

Most state-of-the-art results for such codes rely on synchronization strings, simple yet powerful pseudo-random objects that have proven to be very effective solutions for coping with synchronization errors in various settings. This survey also includes an overview of what is known about synchronization strings and discusses communication settings related to error-correcting codes in which synchronization strings have been applied.

***Index Terms*—Coding for Insertions and Deletions, Synchronization Strings, Error-Correction for Synchronization Errors, List-Decoding.**

I. Introduction

FOLLOWING the inspiring works of Shannon and Hamming a sophisticated and extensive body of research on error-correcting codes has led to a deep and detailed theoretical understanding as well as practical implementations that have helped fuel the Digital Revolution. Error-correcting codes can be found in virtually all modern communication and computation systems. While being remarkably successful in understanding the theoretical limits and trade-offs of reliable communication under substitution errors and erasures, the coding theory literature lags significantly behind when it comes to overcoming errors that concern the timing of communications. In particular, the study of correcting synchronization errors, i.e., symbol insertions and deletions, while initially introduced by Levenshtein in the 60s, has significantly fallen behind our highly sophisticated knowledge of codes for Hamming-type errors, that are symbol substitutions and erasures.

This discrepancy has been well noted in the literature. An expert panel [1] in 1963 concluded: *"There has been one glaring hole in [Shannon's] theory; viz., uncertainties in timing, which I will propose to call time noise, have not been encompassed .... Our thesis here today is that the synchronization problem is not a mere engineering detail, but a fundamental communication problem as basic as detection itself!"* however as noted in a comprehensive survey [2] in 2010: *"Unfortunately, although it has early and often been conjectured that error-correcting codes capable of correcting timing errors could improve the overall performance of communication systems, they are quite challenging to design, which partly explains why a large collection of synchronization techniques not based on coding were developed and implemented over the years."* or as Mitzenmacher puts in his survey [3]: *"Channels with synchronization errors, including both insertions and deletions as well as more general timing errors, are simply not adequately understood by current theory. Given the near-complete knowledge we have for channels with erasures and errors ... our lack of understanding about channels with synchronization errors is truly remarkable."*

However, over the last five years, partially spurred by new emerging application areas, such as DNA-storage [4], [5], [6], [7], [8], [9], significant breakthroughs in our theoretical understanding of error correction methods for insertions and deletions have been made.

This survey focuses on error-correcting codes over finite alphabets that can correct a constant fraction of worst-case insertions and deletions and provides a complete account of the recent progress in this area. Much of this progress has been obtained through synchronization strings, recently introduced, simple yet powerful pseudo-random objects proven to be very effective solutions for coping with synchronization errors in various communication settings. This paper includes streamlined and self-contained proofs for the state-of-the-art code constructions and decoding procedures for both unique-decodable and list-decodable error-correcting codes over large constant alphabets, which are based on synchronization strings. We also provide in-depth discussions of such codes over binary and other fixed (small) alphabets. Lastly, this paper includes an overview of what is known about synchronization strings themselves and discusses other communication settings in which synchronization strings have been successfully applied.

A. Synchronization Errors

Consider a stream of symbols being transmitted through a noisy channel. There are two basic types of errors that we will consider, Hamming-type errors and synchronization errors. *Hamming-type errors* consist of *erasures*, that is, a

Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 15213, USA. E-mails: haeupler@cs.cmu.edu, shahrasbi@cs.cmu.edu. Supported in part by NSF grants CCF-1527110, CCF-1618280, CCF-1814603, CCF-1910588, NSF CAREER award CCF-1750808, a Sloan Research Fellowship, and funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (ERC grant agreement 949272).

symbol being replaced with a special "?" symbol indicating the erasure, and *substitutions* in which a symbol is replaced with any other symbol of the alphabet. We will measure Hamming-type errors in terms of *half-errors*. The wording half-error comes from the realization that, when it comes to code distances, erasures are half as bad as symbol substitutions. An erasure is thus counted as one half-error while a symbol substitution counts as two half-errors. *Synchronization errors* consist of *deletions*, that is, a symbol being removed without replacement, and *insertions*, where a new symbol is added somewhere within the stream.

Synchronization errors are strictly more general and harsher than half-errors. In particular, any symbol substitution, worth two half-errors, can also be achieved via a deletion followed by an insertion. Any erasure can be interpreted as a deletion together with the extra information where this deletion has taken place. This shows that any error pattern generated by k half-errors can also be replicated using k synchronization errors, making dealing with synchronization errors at least as hard as half-errors. The real problem that synchronization errors bring about, however, is that they cause sending and receiving parties to become "out of sync". This easily changes how received symbols are interpreted and makes designing codes or other systems tolerant to synchronization errors an inherently difficult and significantly less well-understood problem.

B. Scope of the Survey and Related Works

The study of coding for synchronization errors was initiated by Levenshtein [10] in 1966 when he showed that Varshamov-Tenengolts codes can correct a single insertion, deletion, or substitution error with an optimal redundancy of almost $\log n$ bits. Ever since, synchronization errors have been studied in various settings. In this section, we specify and categorize some of the commonly studied settings and give a detailed summary of past works within the scope of this survey.

The first important aspect is the noise model. Several works have studied coding for synchronization errors under the assumption of random errors, most notably, to study the capacity of deletion channels, which independently delete each symbol with some fixed probability. In this paper, we exclusively focus on worst-case error models in which correction has to be possible from any (adversarial) error pattern bounded only by the total number of insertions and deletions. We refer to the recent survey (in the same special issue) by Cheraghchi and Ribeiro [11] on capacity results for synchronization channels as well as the surveys by Mitzenmacher [3] and Mercier [2], for an extensive review of the literature on codes for random synchronization errors.

Another angle to categorize the previous work on codes for synchronization error from is the noise regime. In the same spirit as ordinary error-correcting codes, the study of families of synchronization codes has included both ones that protect against a fixed number of synchronization errors and ones that consider error count that is a fixed fraction of the block length. The inspiring work of Levenshtein [10] falls under the first category and is followed by several works designing synchronization codes correcting k errors for specific values of k [12], [13], [14], [15] or with k as a general parameter [16], [17]. In this work, we focus on the second category, i.e., infinite families of synchronization codes with increasing block length that are defined over a fixed alphabet size and can correct from constant-fractions of worst-case synchronization errors.

Furthermore, we mainly focus on codes that can be efficiently constructed and decoded – in contrast to merely existential results. The first such code was constructed in 1990 by Schulman and Zuckerman [18]. They provided an efficient, asymptotically good synchronization code with constant rate and constant distance. In the following, we will give a complete review of the previous work relevant to the scope of this paper.

1) Rate-Distance Trade-Off: One of the main problems in coding theory concerns the question of what the largest achievable communication rate is while protecting from a certain fraction of (synchronization) errors. This question can be studied under the regime that assumes some fixed alphabet of size q, specifically binary alphabets, or an alphabet-free regime that studies the rate achievability when alphabet size can be chosen arbitrarily large but independent of the block length.

For the large alphabet setting, the Singleton bound suggests that no family of codes can correct a δ fraction of deletions, and hence, δ fraction of synchronization errors while achieving a rate strictly larger than $1-\delta$. A series of works by Guruswami *et al*. [19], [20] provides codes that achieve a rate of $\Omega((1-\delta)^5)$ and $1-\tilde{O}(\sqrt{\delta})$ while being able to efficiently recover from a δ fraction of insertions and deletions in high-noise and high-rate regimes respectively. In this paper, we will take a deep dive into synchronization string based code constructions that provide codes that can approach the Singleton bound up to an arbitrarily small additive term over the entire distance spectrum $\delta \in (0,1)$.

For binary alphabet codes, one can show that the optimal achievable rate to protect against a δ fraction of insertions or deletions is $1-O(\delta \log \frac{1}{\delta})$ [10]. Works of Guruswami *et al*. [19], [20] and Haeupler *et al*. [21] present efficient codes with distance δ and rate $1-O\left(\sqrt{\delta}\log^{O(1)}\frac{1}{\delta}\right)$ for sufficiently small δ. Recent works by Cheng *et al*. [22] and Haeupler [23] have achieved codes with rate $1-O(\delta \log^2 \frac{1}{\delta})$.

2) List Decoding: Like error-correcting codes, synchronization codes have been studied under the *list decoding* model where, as opposed to unique decoding, the decoder is expected to produce a list of codewords containing the transmitted codeword as long as the error rate is sufficiently small.

Guruswami and Wang [20] have provided positive-rate binary deletion codes that can be list-decoded from close to $\frac{1}{2}$ fraction of deletions. Haeupler *et al*. [24], [25] gave upper and lower bounds on the maximum achievable rate of list-decodable insertion-deletion codes (or insdel codes for short) over any alphabet size q. Recent works of Wachter-Zeh [26] and Hayashi and Yasunaga [27] have studied list-decoding by providing Johnson-type bounds for synchronization codes that relate the minimum edit-distance of the code to its list

decoding properties. We generally define the edit-distance between two strings as the smallest number of insertions and deletions needed to convert one to another. The bounds presented in [27] show that binary codes by Bukh, Guruswami, and Håstad [28] can be list-decoded from a fraction ≈ 0.707 of insertions. Via a concatenation scheme used in [20] and [19], Hayashi and Yasunaga furthermore made these codes efficient. A recent work of Liu, Tjuawinata, and Xing [29] also derives bounds on list-decoding radius, provides efficiently list-decodable insertion-deletion codes over small alphabets, and gives a Zyablov-type bound for synchronization codes.

3) Error Resilience: As mentioned above, it is known that there exist positive-rate binary deletion codes that are list-decodable from any fraction of errors smaller than $\frac{1}{2}$. Also, there are codes that can list-decode from a fraction ≈ 0.707 of insertions. We will present a recent result from [30] that, for any alphabet size q, precisely identifies the maximal rates of combinations of insertion and deletion errors from which list-decoding is possible.

A similar question can be asked for uniquely-decodable synchronization codes, i.e., what is the largest fraction of errors δ_0 where there exist positive-rate synchronization codes with minimum edit-distance δ_0? For binary alphabets, it is easy to see that $\delta_0 \leq \frac{1}{2}$. However, most resilient binary codes with positive rate to date are ones introduced by Bukh, Guruswami, and Håstad [28] that can correct a $\sqrt{2}-1 \approx 0.4142$ fraction of errors. Determining the optimal error resilience for uniquely-decodable synchronization codes remains an interesting open question. We refer the reader to [11] for a more comprehensive review of past works on the error resilience for synchronization codes.

C. Coding with Synchronization Strings

One commonly studied approach to correct from synchronization errors is to use special symbols or sequences with specific structures as markers or delimiters to keep track of insertions and deletions and realign a transmitted word [31], [32], [33], [34], [35], [36], [37], [38]. In this work, we focus on a very recent form of such technique – indexing with synchronization strings.

Introduced in [39], synchronization strings allow efficient synchronization of streams that are affected by insertions and deletions using an abstract indexing scheme. Essentially, synchronization strings enable compartmentalization of coding against synchronization errors into two steps of (1) realigning the received stream of symbols in a way that guarantees most symbols are in their original position and (2) coding against Hamming-type errors caused by wrong realignments. Synchronization strings have made progress on a wide variety of settings and problems. This survey focuses on code constructions that are based on synchronization strings. Most importantly, we will review the following results.

1) Codes Approaching the Singleton Bound: Synchronization strings enable construction of families of synchronization codes that approach an almost optimal rate-distance trade-off as suggested by the Singleton bound over constant alphabet sizes. In other words, as shown in [39], for any $0 \leq \delta < 1$ and any $\varepsilon > 0$, there exists a family of synchronization codes that can uniquely and efficiently correct any δ fraction of insertions and deletions and achieve a rate of $1-\delta-\varepsilon$. Such codes exist over alphabets of size $\exp(1/\varepsilon)$ which is shown in [40] to be the asymptotically optimal alphabet size for a code with such properties.

2) Near-Linear Time Codes: We then present an improvement from [41] over the result just described that modifies the construction and decoding in a way that enables near-linear time decoding. Two main ingredients are used to achieve this improvement: (1) generalizations of synchronization strings and their fast construction methods introduced in [40], and (2) a fast indexing scheme for edit-distance computation from [42]. For any n and $\varepsilon > 0$, [42] gives string I of length n over an alphabet of size $|\Sigma| = O_\varepsilon(1)$ which enables fast approximation of the edit distance in the following way: Let $S \in \Sigma'^n$ be another string of length n over some other alphabet Σ'. If one concatenates S and I, symbol-by-symbol, to obtain the string $S \times I \in (\Sigma \times \Sigma')^n$, then edit distance from any other string $S' \in (\Sigma \times \Sigma')^*$ to $S \times I$ can be approximated within a multiplicative factor of $1+\varepsilon$ in near-linear time.

3) List Decoding for Insertions and Deletions: We then proceed to present a recent result on list-decodable synchronization codes. Using a similar synchronization string-based approach, [24] shows that for every $0 \leq \delta < 1$, every $0 \leq \gamma < \infty$ and every $\varepsilon > 0$ there exist a family of codes with rate $1-\delta-\varepsilon$, over an alphabet of constant size $q = O_{\delta,\gamma,\varepsilon}(1)$ that are list-decodable from a δ-fraction of deletions and a γ-fraction of insertions. This family of codes are efficiently decodable and their decoding list size is sub-logarithmic in terms of the code's block length. We stress that the fraction of insertions can be arbitrarily large (even more than 100%) and the rate is independent of this parameter.

4) Optimal Error Resilience for List Decoding: Finally, we review a result by Guruswami *et al*. [30] that, using a code concatenation scheme for synchronization codes with codes from [28] and [24], exactly identifies the maximal fraction of insertions and deletions that can be tolerated by q-ary list-decodable codes with non-vanishing information rate. This includes efficient binary codes that can be list-decoded from any δ fraction of deletions and γ fraction of insertions as long as $2\delta + \gamma < 1$. One can show that list decoding is not possible for any family of codes achieving positive rates for any error fraction out of this region. Guruswami *et al*. [30] have generalized this result to alphabets of size q and identified the feasibility region for (γ, δ) as a more complex region with a piece-wise linear boundary.

D. Organization of the Paper

In Section II, we will provide proofs for claims presented in Section I-C by formally introducing indexing based code constructions and giving a minimal introduction to pseudo-random strings used for indexing. In Section III, we discuss several pseudo-random string properties, their constructions, their repositioning algorithms and the decoding properties

that they enable once used to construct codes. We then mention applications of synchronization strings and related string properties in other communication problems such as coding for block errors and interactive communication under synchronization errors in Section IV.

II. Code via Indexing

In this section, we explain the construction of codes stated in Section I-C. We start with a self-contained simplified proof of Singleton bound approaching codes presented in Section I-C1 that encapsulates the major ideas behind synchronization string-based code constructions while avoiding unnecessary details.

A. Approaching the Singleton Bound: Technical Warm-up

We start by defining the notion of *ε-self-matching strings* that satisfy a weaker property than synchronization strings but can be used in a similar fashion to construct synchronization codes.

Definition II.1. *String $S \in \Sigma^n$ is ε-self-matching if it contains no two identical non-aligned subsequences of length $n\varepsilon$ or more, i.e., there exist no two sequences $a_1, a_2, \ldots, a_{\lfloor n\varepsilon \rfloor}$ and $b_1, b_2, \ldots, b_{\lfloor n\varepsilon \rfloor}$ where for all is $a_i \neq b_i$ and $S[a_i] = S[b_i]$.*

1) Pseudo-random Property: We first point out that random strings over an alphabet of size $\Omega(\varepsilon^{-2})$ satisfy ε-self-matching property with high probability. Note that the probability of two given non-aligned subsequences of length $n\varepsilon$ in a random string over alphabet Σ being identical is $\frac{1}{|\Sigma|^{n\varepsilon}}$. Also, there are no more than $\binom{n}{n\varepsilon}^2$ pairs of such subsequences. Therefore, by the union bound, the probability of such random string satisfying ε-self-matching property is $\binom{n}{n\varepsilon}^2 \frac{1}{|\Sigma|^{n\varepsilon}} \leq \left(\frac{ne}{n\varepsilon}\right)^{2n\varepsilon} \frac{1}{|\Sigma|^{n\varepsilon}} = \left(\frac{e^2}{|\Sigma|\varepsilon^2}\right)^{n\varepsilon}$ and thus, if $|\Sigma| = \Omega(\varepsilon^{-2})$, the random string would satisfy the ε-self-matching property with high probability.

2) Indexing Scheme: Consider a communication channel where a stream of n message symbols are communicated from the sender to the receiver and assume that the communication may suffer from up to $n\delta$ adversarial insertions or deletions for some $0 \leq \delta < 1$. We introduce a simple indexing scheme that will be used to construct synchronization codes. Let $m_1, m_2, \ldots, m_n$ represent the message symbols that the sender wants to get to the receiver and $s_1, s_2, \ldots, s_n$ be some ε-self-matching string that the sender and the receiver have agreed upon beforehand. To communicate its message to the receiver, we have the sender send the sequence $(m_1, s_1), (m_2, s_2), \ldots, (m_n, s_n)$ through the channel. We will refer to this sequence as m indexed by s and denote it by $m \times s$. Note that in this setting a portion of the channel alphabet is designated to the ε-self-matching string and thus, does not contain information. This portion will be used to reposition the message symbols on the receiving end of the communication as we will describe in the next section.

3) Repositioning (Decoding): We now show that, having the indexing scheme described above, the receiver can correctly identify the positions of most of the symbols it receives. Let us denote the sequence of symbols arriving at the receiving end by $(m'_1, s'_1), (m'_2, s'_2), \ldots, (m'_{n'}, s'_{n'})$. We show the following.

Lemma II.2. *There exists an algorithm for the receiving party that, having $(m'_1, s'_1), \ldots, (m'_{n'}, s'_{n'})$ and $s_1, \ldots, s_n$, guesses the* position *of all received symbols in the sent string such that positions of all but $O(n\sqrt{\varepsilon})$ of the symbols that are not deleted in the channel are guessed correctly. This algorithm runs in $O_\varepsilon(n^2)$ time.*

Note that if no error occurs, the receiver expects the index portion of the received symbols to be similar to the ε-self-matching string s. Having this observation, we present the decoding procedure in Algorithm 1. The decoding algorithm calculates the longest common subsequence (LCS) between the synchronization string, s, and the index portion of the received string, s'. It then assigns each of the symbols from the received string that appear in the common subsequence to the position of the symbol from s that corresponds to it under the common subsequence. The algorithm repeats this procedure $1/\sqrt{\varepsilon}$ times and after each round eliminates received symbols whose positions are guessed.

Algorithm 1 Insertion-Deletion Decoder

Input: s, $(m'_1, s'_1), \cdots, (m'_{n'}, s'_{n'})$

$L = [s'_1, s'_2, \cdots, s'_{n'}]$
for $i = 1$ to n' **do**
  $Position[i] \leftarrow$ `Undetermined`
end for
for $i = 1$ to $\frac{1}{\sqrt{\varepsilon}}$ **do**
  Compute LCS(s, L)
  for all Corresponding $s[i]$ and $L[j]$ in LCS(s, L) **do**
    $Position[j] \leftarrow i$
  end for
  Remove all elements of LCS(s, L) from L
end for

Output: $Position$

Proof of Lemma II.2. Clearly, Algorithm 1 takes quadratic time as it mainly runs $O_\varepsilon(1)$ instances of LCS computation over strings of length $O(n)$.

To prove the correctness guarantee, we remark that there are two types of incorrect guesses for symbols that are not deleted by the adversary and bound the number of incorrect guesses of each type.

I) *The position of the received symbol remains* `Undetermined` *by the end of the algorithm:* Note that if by the end of the algorithm there are k original symbols–i.e., symbols that are originally sent by the sender and not inserted by the adversary–that have undetermined positions, then the remainder of L after $1/\sqrt{\varepsilon}$ rounds has a common subsequence of size k with s. This implies that, in each round of the for

loop, $|\text{LCS}(s, L)| \geq k$. Note the total size of these LCSs cannot exceed the initial size of L that is n'. Therefore, $k \cdot \frac{1}{\sqrt{\varepsilon}} \leq n' \leq 2n \Rightarrow k \leq 2\sqrt{\varepsilon}n$.

II) *The position of the received symbol is incorrectly guessed in one recurrence of the for loop:* We claim that the number of such wrong assignments in each round of the for loop is no more than $n\varepsilon$. Let $s[i]$ and $L[j]$ be corresponding elements under $\text{LCS}(s, L)$ in Line 7 while the received symbol that $L[j]$ identifies is the i'th symbol sent by the sender. This implies that $s[i] = L[j] = s[i']$. If there are more than $n\varepsilon$ such incorrect guesses in one LCS computation, we have $n\varepsilon$ such pairs of identical symbols in s that constitute a self-matching of size $n\varepsilon$ in s and violate the assumption of s being an ε-self-matching string. Therefore, overall there are no more than $\frac{1}{\sqrt{\varepsilon}} \cdot n\varepsilon = n\sqrt{\varepsilon}$ incorrect determination of the original positions of received symbols. □

4) Codes Approaching the Singleton Bound: We now use the discussions on ε-self-matching strings and Lemma II.2 to construct efficient synchronization codes that can approach the Singleton bound.

Theorem II.3. *For any $\varepsilon > 0$, $\delta \in (0, 1)$, and sufficiently large n, there exists an encoding map $E : \Sigma^k \to \Sigma^n$ and a decoding map $D : \Sigma^* \to \Sigma^k$, such that, if $\text{ED}(E(m), x) \leq \delta n$ then $D(x) = m$. Further, the rate is $\frac{k}{n} > 1-\delta-\varepsilon$, $|\Sigma| = \exp(1/\varepsilon)$, and E and D are explicit and can be computed in linear and quadratic time in n.*

We use $\text{ED}(x, y)$ to denote the edit distance between x and y. Note that the indexing scheme from Section II-A2 and Lemma II.2 essentially gives a way to reduce insertions and deletions to symbol substitutions and erasures at the cost of designating a portion of the message symbols to an ε-self-matching string. More precisely, with the indexing scheme from Section II-A2 in place, a receiver can use Algorithm 1 to guess the position of the symbols it receives in the sent message and rearrange them to recover the message sent by the sender.

Let $\tilde{m}$ denote the recovered message and $Position$ denote the output of Algorithm 1. More precisely, for any $1 \leq i \leq n$, the decoder sets $\tilde{m}[i] = j$ if only for one value of j, $Position[j] = i$. If there are zero or multiple received symbols that are guessed to be at position i, the decoder simply decides $\tilde{m}[i] = ?$.

We claim that $\tilde{m}$ is different from m by no more than $n(\delta + 12\sqrt{\varepsilon})$ half-errors. Note that if an adversary applies no errors and Algorithm 1 guesses the positions perfectly, $\tilde{m} = m$. In the following steps, we add these imperfections and see the effect in the Hamming distance between m and $\tilde{m}$.

- Each deleted symbol turns a detection in $\tilde{m}$ to a ? and, therefore, adds one half-error to the Hamming distance between m and $\tilde{m}$.
- Each inserted symbol can either turn a detection in $\tilde{m}$ to a ? or a ? to an incorrect value. Therefore, each insertion also adds one half-error to the Hamming distance between m and $\tilde{m}$.
- Each incorrectly guessed symbol can also change up to two symbols in $\tilde{m}$ and therefore increase the Hamming distance between m and $\tilde{m}$ by up to four.

This implies that the m and $\tilde{m}$ are far apart by no more than $n(\delta+12\sqrt{\varepsilon})$. Having this reduction, we derive codes promised in Theorem II.3 by taking the following near-MDS codes from [43] and indexing their codewords with a self-matching string.

Theorem II.4 (Guruswami and Indyk [43, Theorem 3])**.** *For every r, $0 < r < 1$, and all sufficiently small $\varepsilon > 0$, there exists an explicitly specified family of GF(2)-linear (also called additive) codes of rate r and relative distance at least $(1 - r - \varepsilon)$ over an alphabet of size $2^{O(\varepsilon^{-4}r^{-1}\log(1/\varepsilon))}$ such that codes from the family can be encoded in linear time and can also be (uniquely) decoded in linear time from a fraction e of errors and s of erasures provided $2e + s \leq (1 - r - \varepsilon)$.*

Proof of Theorem II.3. Let C be a code from Theorem II.4 with relative distance $\delta_C = \delta + \frac{\varepsilon}{3}$ and rate $1 - \delta_C - \varepsilon_C$ for $\varepsilon_C = \frac{\varepsilon}{3}$ and S be an ε_S-self-matching string with parameter $\varepsilon_S = \frac{\varepsilon^2}{36^2}$. We construct code C' by simply taking the code C and indexing each codeword of it with S. We claim that the resulting code satisfies the properties promised in the statement of the theorem.

We start with showing the decoding guarantee by describing the decoder. Note that a decoder can use the procedure described in Algorithm 1 to use the index portion of codewords to reconstruct the codeword by up to a $\delta+12\sqrt{\varepsilon'} = \delta+\frac{\varepsilon}{3} = \delta_C$ fraction of half-errors. The decoder then simply feeds the resulting string into the decoder of C to fully recover the original string. The encoding and decoding complexities of C' follow from the fact that C is encodable and decodable in linear time and that Algorithm 1 runs in quadratic time.

We finish the proof with proving the rate guarantee. Note that $\Sigma_{C'} = \Sigma_C \times \Sigma_S$.

$$\begin{aligned} r_{C'} &= \frac{|C'|}{n \log |\Sigma_{C'}|} = \frac{|C|}{n \log \left(|\Sigma_C| \times |\Sigma_S|\right)} \\ &= r_C \cdot \frac{\log |\Sigma_C|}{\log |\Sigma_C| + \log |\Sigma_S|} = \frac{r_C}{1 + \frac{\log |\Sigma_S|}{\log |\Sigma_C|}} \end{aligned} \quad (1)$$

Note that the discussion in Section II-A1 implies that S can be over an alphabet of size $|\Sigma_S| = O(\varepsilon^{-2})$ and Theorem II.4 gives that $\log |\Sigma_C| = \omega(\varepsilon^{-4} \log 1/\varepsilon)$. Thus, $\frac{\log |\Sigma_S|}{\log |\Sigma_C|} = O(\varepsilon^{-4})$, which plugged in (1) implies that $r_{C'} \geq \frac{r_C}{1+O(\varepsilon^4)} \geq r_C - \frac{\varepsilon}{3}$. Therefore, since the rate of code C is $r_C = 1 - \delta_C - \varepsilon_C = 1 - \delta - \frac{2}{3}\varepsilon$, the rate of the code C' is at least $r_{C'} = 1 - \delta - \frac{2}{3}\varepsilon - \frac{\varepsilon}{3} = 1 - \delta - \varepsilon$. □

Note that the alphabet size of the codes from Theorem II.3 is exponentially large in terms of ε^{-1}. This is in sharp contrast to the Hamming error setting where there are codes known that can get ε close to unique decoding capacity with alphabets of polynomial size in terms of $1/\varepsilon$. While large alphabet sizes might seem as an intrinsic weakness of the indexing-based code constructions, it turns out that an exponentially large alphabet size is actually necessary. We present the following theorem from [24] that shows any such code requires exponentially large alphabet size in terms of $\exp(\varepsilon^{-1})$.

Theorem II.5. *There exists a function $f : (0,1) \to (0,1)$ such that for every $\delta, \varepsilon > 0$, every family of insertion-deletion codes of rate $1 - \delta - \varepsilon$ that can be uniquely decoded from δ-fraction of synchronization errors must have alphabet size $q \geq \exp\left(\frac{f(\delta)}{\varepsilon}\right)$.*

Proof Sketch. For simplicity, assume that $\delta = \frac{d}{q}$ for some integer d. Consider a code of block length n and an adversary that always deletes all occurrences of the d least frequent symbols. With such an adversary, the string received on receiver's side will be a string of length $n(1-\delta)$ over an alphabet of size $q - d = q(1-\delta)$. This means that there are a total of $M = \binom{q}{q-d}(q-d)^{n(1-\delta)}$ possible strings that may arrive at the receiver's end which implies that the rate of any such code is no more than

$$\frac{\log M}{n \log q} = (1-\delta)\left(1 + \frac{\log(1-\delta)}{\log q}\right) + o(1).$$

Therefore, to achieve a rate of $1 - \delta - \varepsilon$,

$$1 - \delta - \varepsilon \leq (1-\delta)\left(1 + \frac{\log(1-\delta)}{\log q}\right)$$

$$\Rightarrow \quad (1-\delta)\frac{\log \frac{1}{1-\delta}}{\log q} \leq \varepsilon \Rightarrow q \geq e^{(1-\delta)\log \frac{1}{1-\delta}}$$

For the general case where δq is not necessarily an integer, a similar, more careful argument proves the theorem. (See [24].) □

The alphabet reduction idea used in the proof of Theorem II.5 shows that deletions, in addition to reducing the information by eliminating symbols, reduce the information by essentially decreasing the information content of surviving symbols; suggesting that designating a part of each symbol to synchronization strings is not a waste of information. A similar alphabet reduction argument is used in [24], [25] to derive strong upper-bounds on the zero-error list-decoding capacity of adversarial insertion-deletion channels.

B. Near-Linear Time Codes

In Section II-A, we presented a way to construct synchronization codes that approach the Singleton bound by taking a near-MDS error-correcting code and indexing its codewords with self-matching strings. In this section, we explain how the decoding complexity of such codes can be reduced to near-linear time.

The main idea is to replace the ε-self matching string with one that satisfies a stronger pseudo-random property that allows for a near-linear time repositioning algorithm. We will thoroughly explain the construction of such a string and its repositioning algorithm in Section III-F. We forward reference the properties of this string in the following theorem and defer the details to Section III-F.

Theorem II.6 (Theorem III.11 with $\varepsilon_I = \frac{2\varepsilon}{9}, \varepsilon_s = \frac{\varepsilon^2}{18}, K = \frac{6}{\varepsilon}, \gamma = 1$)**.** *For any $\varepsilon > 0$, there exist strings of any length n over an alphabet of size $\exp\left(\frac{\log(1/\varepsilon)}{\varepsilon^3}\right)$ that, if used as an index in a synchronization channel with δ fraction of errors, enables a repositioning in $O_\varepsilon(n\mathsf{poly}(\log n))$ time which guarantees no more than $n\varepsilon$ incorrect guesses.*

Using these strings in the code construction, the following can be achieved.

Theorem II.7. *For any $\varepsilon > 0$ and $\delta \in (0,1)$, and sufficiently large n, there exists an encoding map $E : \Sigma^k \to \Sigma^n$ and a decoding map $D : \Sigma^* \to \Sigma^k$, such that, if $\mathrm{ED}(E(m), x) \leq \delta n$ then $D(x) = m$. Further, $\frac{k}{n} > 1 - \delta - \varepsilon$, $|\Sigma| = \exp\left(\varepsilon^{-4}\log(1/\varepsilon)\right)$, and E and D are explicit and can be computed in linear and near-linear time in terms of n respectively.*

Proof Sketch. To construct such codes with a given ε, we use strings from Theorem II.6 with parameter $\frac{\varepsilon}{4}$ as an index string. We then take code C from [43] as a code with distance $\delta_C = \delta + \frac{\varepsilon}{2}$ and rate $1 - \delta_C - \frac{\varepsilon}{4}$ over an alphabet of size $|\Sigma_C| \geq |\Sigma_S|^{4/\varepsilon}$. Note that $|\Sigma_S| = \exp\left(\frac{\log(1/\varepsilon)}{\varepsilon^3}\right)$, therefore, the choice of $|\Sigma_C|$ is large enough to satisfy the requirements of [43]. C is also encodable and decodable in linear time.

With this choice of C and S, the same analysis as in Section II-A shows that the resulting synchronization code can be encoded in linear time, be decoded in $O_\varepsilon(n\mathsf{poly}(\log n))$ time, corrects from any δn insertions and deletions, achieves a rate of $\frac{R_C}{1+\frac{\log|\Sigma_S|}{\log|\Sigma_C|}} \geq \frac{1-\delta-3\varepsilon/4}{1+\varepsilon/4} \geq 1 - \delta - \varepsilon$, and is over an alphabet of size $\exp\left(\frac{\log(1/\varepsilon)}{\varepsilon^4}\right)$. □

C. List Decoding: High Rate Codes

In this section, we review the results described in Section I-C3 that, for every $0 \leq \delta < 1$, every $0 \leq \gamma < \infty$ and every $\varepsilon > 0$, gives list-decodable codes with rate $1 - \delta - \varepsilon$, constant alphabet (so $q = O_{\delta,\gamma,\varepsilon}(1)$), and sub-logarithmic list sizes. Furthermore, these codes are accompanied by efficient (polynomial time) decoding algorithms. We stress that the fraction of insertions can be arbitrarily large (more than 100%), and the rate is independent of this parameter. Here is a formal statement of the result from [24].

Theorem II.8. *For every $0 < \delta, \varepsilon < 1$ and $\gamma > 0$, there exist a family of list-decodable insertion-deletion codes that can protect against δ-fraction of deletions and γ-fraction of insertions and achieves a rate of at least $1 - \delta - \varepsilon$ or more over an alphabet of size $\left(\frac{\gamma+1}{\varepsilon^2}\right)^{O\left(\frac{\gamma+1}{\varepsilon^3}\right)} = O_{\gamma,\varepsilon}(1)$. These codes are list-decodable with lists of size $L_{\varepsilon,\gamma}(n) = \exp\left(\exp\left(\exp\left(\log^* n\right)\right)\right)$, and have polynomial time encoding and decoding complexities.*

The construction of these codes is similar to the ones from Theorem II.7 except that the error-correcting code used in the construction is replaced with a high-rate list-recoverable code. A code C given by the encoding function $\mathcal{E} : \Sigma^{nr} \to \Sigma^n$ is called to be (α, l, L)-list recoverable if for any collection of n sets $S_1, S_2, \ldots, S_n \subseteq \Sigma$ each of size l or less, there are at most L codewords for which more than αn elements appear in the list that corresponds to their position, i.e.,

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

The main idea is to use indexes and the repositioning algorithm from Algorithm 1 to come up with a list of candidate symbols for each position of the original message and then

feed these lists to the decoder of the list-recoverable code. To prove Theorem II.8, the following family of list-recoverable codes from [44] is utilized.

Theorem II.9 (Hemenway *et al*. [44, Theorem A.7])**.** *Let q be an even power of a prime, and choose $l, \epsilon > 0$, so that $q \geq \epsilon^{-2}$. Choose $\rho \in (0,1)$. There is an $m_{min} = O(l \log_q(l/\epsilon)/\epsilon^2)$ so that the following holds for all $m \geq m_{min}$. For infinitely many n (all n of the form $q^{e/2}(\sqrt{q}-1)$ for any integer e), there is a deterministic polynomial-time construction of an F_q-linear code $C : \mathbb{F}_{q^m}^{\rho n} \to \mathbb{F}_{q^m}^{n}$ of rate ρ and relative distance $1-\rho-O(\epsilon)$ that is $(1-\rho-\epsilon, l, L)$-list-recoverable in time $\mathsf{poly}(n, L)$, returning a list of codewords that are all contained in a subspace over $\mathbb{F}_q$ of dimension at most $\left(\frac{l}{\epsilon}\right)^{2^{\log^*(mn)}}$; implying that $L \leq q^{(l/\epsilon)^{2^{\log^*(mn)}}}$.*

Proof of Theorem II.8. By setting parameters $\rho = 1-\delta-\frac{\varepsilon}{2}$, $l = \frac{2(\gamma+1)}{\varepsilon}$, and $\epsilon = \frac{\varepsilon}{4}$ in Theorem II.9, one can obtain a family of codes $\mathcal{C}$ that achieves rate $\rho = 1-\delta-\frac{\varepsilon}{2}$ and is (α, l, L)-recoverable in polynomial time for $\alpha = 1-\delta-\varepsilon/4$ and some $L = \exp(\exp(\exp(\log^* n)))$ (by treating γ and ε as constants). Such a family of codes can be found over an alphabet $\Sigma_{\mathcal{C}}$ of size $q = (l/\epsilon)^{O(l/\epsilon^2)} = \left(\frac{\gamma+1}{\varepsilon^2}\right)^{O\left(\frac{\gamma+1}{\varepsilon^3}\right)} = O_{\gamma,\varepsilon}(1)$ or infinitely many integer numbers larger than q.

We index the codewords of this code with an $\varepsilon_s = \frac{\varepsilon^2}{64(1+\gamma)}$ self-matching string S. We now show that these codes satisfy the list-decoding properties presented in the statement of the theorem.

The decoder starts with guessing the positions for the symbols it receives using a repositioning algorithm similar to Algorithm 1 with two minor differences:

1) Instead of reconstructing the original string with guessed positions and ?s, the decoder compiles a list for each position containing all received symbols that have been guessed to be in that position.
2) The algorithm repeats the procedure of calculating LCS and adding elements to the lists for a total of $K = \frac{8(1+\gamma)}{\varepsilon}$ times.

A similar analysis to the one for Algorithm 1 shows that the count of the lists that do not contain the original symbol that corresponds to their position is no more than $n\left(\delta + \frac{1+\gamma}{K} + K\varepsilon_s\right) = n(\delta + \varepsilon/4)$.

Then, the decoding algorithm feeds these lists into the decoder of the list-recoverable code from $\mathcal{C}$ to obtain a list of size L of potential original messages. Since the parameter α was chosen to be $1-\delta-\varepsilon/4$, the output list is guaranteed to contain the original message.

The rate of the resulting family of codes is $\frac{1-\delta-\varepsilon/2}{1+\log|\Sigma_S|/\log|\Sigma_{\mathcal{C}}|}$ which, by taking $|\Sigma_{\mathcal{C}}|$ large enough in terms of ε, is larger than $1-\delta-\varepsilon$. As $\mathcal{C}$ is encodable and decodable in polynomial time, the encoding and decoding complexities of the indexed code will be polynomial as well. □

We remark that the self-matching string in the construction of list-decodable synchronization codes from Theorem II.8 can be replaced with the near-linear time repositionable indexes of Theorem II.6. This would reduce the time complexity of the repositioning subroutine in the decoding algorithm to near-linear time. Therefore, it would allow one to improve the decoding complexity of these codes upon discovery of high-rate list-recoverable codes with faster decoders, potentially to near-linear time. A recent work of Kopparty *et al*. [45] has broken this barrier and offers list-recoverable tensor codes with a deterministic $n^{1+o(1)}$ time decoding. Using such codes in a similar scheme would yield a near-linear time list-decodable family of codes with similar properties as of Theorem II.8 albeit over alphabet sizes that grow in terms of the block length of the code.

D. List Decoding: Optimal Resilience via Concatenation

In this section, we discuss the result presented in Section I-C4 that fully characterizes error resilience for list-decodable synchronization codes. More precisely, [30] exactly identifies the maximal fraction of insertion and deletion errors tolerable by q-ary list-decodable codes with non-vanishing rate.

We start by describing the result for binary codes. Note that no positive-rate code can be list-decoded from a $\delta = \frac{1}{2}$ fraction of deletions as an adversary can simply delete all instances of the less frequent symbol. Similarly, no positive-rate code can be list-decoded from a fraction $\gamma = 1$ of insertions since any string of length n can be turned into $(01)^n$ with n insertions. A simple time-sharing argument would show that an adversary that can apply any combination of δ fraction of deletions and γ-fraction of insertions that satisfy $\gamma + 2\delta = 1$ can make the list-decoding impossible. The following theorem from [30] shows the existence of positive-rate list-decodable codes otherwise.

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

This result is generalized for larger alphabets in [30]. However, the feasibility region for larger alphabet sizes is more complex. We start with showing that list-decoding is impossible for several points (γ, δ) that lie on a quadratic curve. This implies a piece-wise linear outer-bound for the resilience region.

Theorem II.11. *For any alphabet size q and any $i = 1, 2, \cdots, q-1$, no positive-rate q-ary infinite family of insertion-deletion codes can list-decode from $\delta = \frac{q-i}{q}$ fraction of deletions and $\gamma = \frac{i(i-1)}{q}$ fraction of insertions.*

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

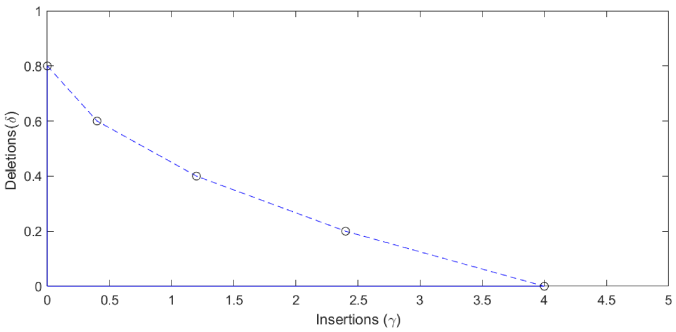

Fig. 1. Feasibility region for $q = 5$.

It is shown in [30] that this is indeed the error resilience region for list-decoding.

Theorem II.12. *For any positive integer $q \geq 2$, let F_q be defined as the concave polygon over vertices $\left(\frac{i(i-1)}{q}, \frac{q-i}{q}\right)$ for $i = 1, \cdots, q$ and $(0,0)$. (An illustration for $q = 5$ is presented in Fig. 1). F_q does not include the border except the two segments $[(0,0),(q-1,0))$ and $[(0,0),(0,1-1/q))$. Then, for any $\varepsilon > 0$ and sufficiently large n, there exists a family of q-ary codes that, as long as $(\gamma,\delta) \in (1-\varepsilon)F_q$, are efficiently L-list decodable from any δn deletions and γn insertions where n denotes the block length of the code, $L = O(\exp(\exp(\exp(\log^* n))))$, and the code achieves a positive rate of $\exp\left(-\frac{1}{\varepsilon^{10}}\log^2\frac{1}{\varepsilon}\right)$.*

We give a high-level description of the steps taken in the proof of Theorem II.10. The proof of Theorem II.12 follows the same blueprint but is more technically involved. For a formal proof of both theorems, we refer the reader to [30].

Theorem II.10 is achieved via two main ingredients. The first is a simple new concatenation scheme for list-decodable synchronization codes which can be used to boost the rate of insdel codes. The second component is a technically intricate proof of the list-decoding properties of the Bukh-Ma codes [46] which have good distance properties but a small sub-constant rate.

(I) Concatenating List-Decodable Synchronization Codes: The first ingredient is a simple but powerful framework for constructing list-decodable insertion-deletion codes via code concatenation. Recall that code concatenation comprises the encoding of an *outer* code C_{out} with an *inner* code C_{in} whose size equals the alphabet size of C_{out}.

In this approach, the outer code C_{out} is chosen to be a list-decodable insdel code C_{out} over an alphabet whose size is some large function of $1/\varepsilon$, has a constant rate, and is capable of tolerating a huge number of insertions. Such a code is introduced in [24] and presented in Theorem II.8.

The inner code C_{in} is chosen to be a list-decodable insdel code over the binary alphabet (or desired alphabet of size q for Theorem II.12), which has non-trivial list decoding properties for the desired fractions γ and δ of insertions and deletions. Most notably, the concatenation framework requires the inner code to be chosen from a family of good list-decodable insdel codes *with an arbitrarily large number of codewords*, and a list-size bounded by some fixed function of $1/\varepsilon$. The codes of Bukh and Ma [46] are shown to satisfy these properties and are used in [30] as the inner code.

We show that even if C_{in} has an essentially arbitrarily bad sub-constant rate and is not efficient, the resulting q-ary insdel code does have constant rate, and can also be efficiently list-decoded from the same fraction of insertions and deletions as C_{in}. For the problem considered in this paper, this framework essentially provides efficiency of codes for free.

The encoding is straightforwardly done by the standard concatenation procedure. The decoding procedure on the other hand, is considerably simpler than similar schemes introduced in earlier works [18], [19], [20], [28]. The decoding is done by (i) list-decoding a sliding substring of the received string using the inner code C_{in}, (ii) creating a single string from the symbols in these lists, and (iii) using the list-decoding algorithm of the outer code on this string (viewed as a version of the outer codeword with some number of deletions and insertions).

The main driving force behind why this simplistic sounding approach actually works is a judicious choice of the outer code C_{out}. Specifically, these codes can tolerate a very large number of insertions. This means that the many extra symbols coming from the list-decodings of the inner code C_{in} and the choice of the (overlapping) sliding intervals does not disrupt the decoding of the outer code. Further, as mentioned above, the list size of the inner code only depends on ε and is independent of the size of the code. This is a crucial property for this concatenation scheme as the following order is used to choose the parameters of C_{in} and C_{out}. Having the parameter ε, the fraction of insertions that the outer code needs to protect against is determined. This would dictate the alphabet size of the outer code and subsequently, the block length or the size of the inner code.

(II) Analyzing the List-Decoding Properties of Bukh-Ma Codes: For the inner code in the concatenation scheme described above, we use a simple family of codes introduced by Bukh and Ma [46], which consist of strings $(0^r\ 1^r)^{\frac{n}{2r}}$ that oscillate between 0s and 1s with different frequencies. (Below we will refer to r as the *period*, and $1/r$ should be thought of as the *frequency* of alternation.) It is shown in [30] that these codes satisfy the following properties.

Theorem II.13. *For any $\varepsilon > 0$ and sufficiently large n, let $C_{n,\varepsilon}$ be the following Bukh-Ma code:*

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

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

In order to prove Theorem II.13, [30] first introduces a new correlation measure which expresses how close a string is to any given frequency (or Bukh-Ma codeword). Using this measure, we want to show that it is impossible to have a single string v which is more than ε-correlated with more than $\Theta_\varepsilon(1)$ frequencies. Loosely speaking, the parameter ε in ε-correlation indicates how close the term $2D + I$ is to 1 where D denotes

the number of deletions and I denotes the number of insertions required to convert the string v into some Bukh-Ma codeword when picking the set of insertions and deletions that minimizes $2D + I$.

The proof technique utilized by [30] is somewhat reminiscent of the one used to establish the polarization of the martingale of entropies in the analysis of polar codes [47], [48]. In more detail, [30] recursively sub-samples smaller and smaller nested substrings of v, and analyzes the expectation and variance of the bias between the fraction of 0's and 1's in these substrings. More precisely, it orders the run lengths $r_1, r_2, \ldots$ that are ε-correlated with v in decreasing order and first samples a substring v_1 with $r_1 \gg |v_1| \gg r_2$ from v. While the expected zero-one bias in v_1 is the same as in v, [30] shows that the variance of this bias is a strictly increasing function in the correlation with $(0^{r_1}1^{r_1})^{\frac{n}{2r_1}}$. Intuitively, v cannot be too uniform on a scale of length $|v_1|$ if it is correlated with r_1.

In other words, if v is ε-correlated with r_1, the sampled substring v_1 will land in a part of v which is either similar to one of the long stretches of zeros in v or in a part which is similar to a long stretch of ones in v, resulting in some positive variance in the bias of v_1. Furthermore, because the scales $r_2, r_3, \ldots$ are so much smaller than v_1, this sub-sampling of v_1 preserves the correlation with these scales intact, at least in expectation.

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

III. Synchronization Strings

In this section, we discuss synchronization strings introduced in [39] and review their combinatorial properties and applications. We also overview extensions and enhancements made to synchronization strings from [49], [21], [40], [24], [42].

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

In this definition, ED represents the edit distance function and $S[x,y)$ denotes a substring of S starting from position x and ending at position $y-1$. We use similar notations $S[x,y]$, $S(x,y]$, and $S(x,y)$ to denote substrings of S where $(,)$ and $[,]$ denote the exclusion and inclusion of the starting/end point of the interval respectively.

In simpler terms, the ε-synchronization property is a pseudo-random property that requires all neighboring substrings of the string to be far apart under the edit distance metric. It is shown in [39] that ε-synchronization is not only a strictly stronger property than the ε-self-matching property but also a hereditary extension of it. More precisely, if all substrings of a string satisfy the $\frac{\varepsilon}{2}$-self matching string property, then the string itself is an ε-synchronization string.

A. Existence

It is shown in [39] that, similar to self matching strings, arbitrarily long ε-synchronization strings exist over alphabets whose size is independent of the string length. More precisely, [39] shows the existence of arbitrarily long strings over an alphabet of size $O(\varepsilon^{-2})$ that satisfy the ε-synchronization property for pairs of neighboring substrings of total length $\frac{1}{\varepsilon^2}$ or more by utilizing the Lovász's local lemma to show that the probability of such an event for a random string is non-zero. Indexing such a string with a string formed by repetitions of $1, 2, \cdots, \varepsilon^{-2}$ ensures the ε-synchronization property over smaller substrings and gives an ε-synchronization string over an alphabet of $O(\varepsilon^{-4})$ size. With a non-uniform sample space, [49] utilizes the Lovász's local lemma in the same manner to reduce the alphabet size to $O(\varepsilon^{-2})$ and gives the following.

Theorem III.2. *For any $\varepsilon \in (0,1)$, there exists an alphabet Σ of size $O(\varepsilon^{-2})$ so that for any $n \geq 1$, there exists an ε-synchronization string of length n over Σ.*

Extremal Properties: We would like to add a brief remark regarding extremal questions that are raised by the definition of the synchronization string property. One interesting question is what is the minimal function of ε as alphabet size for which Theorem III.2 holds. It has been shown in [49] that any such alphabet has to be of size $\Omega(\varepsilon^{-3/2})$. This leaves us with the open question of where the minimal alphabet size lies between $\Omega(\varepsilon^{-3/2})$ and $O(\varepsilon^{-2})$.

A similar question can be asked for non-specific values of ε, i.e., what is the smallest alphabet size over which arbitrarily long ε-synchronization strings exist for any $\varepsilon < 1$. It is easy to observe that any binary string of length 4 or more contains two identical neighboring substrings. Also, it has been shown that arbitrarily long $\frac{11}{12}$-synchronization strings exist over an alphabet of size four [49]. This leaves the open question of whether arbitrarily long synchronization strings exist over a ternary alphabet or not.

B. Online Decoding for Synchronization Strings

In this chapter, we introduce an online repositioning algorithm for synchronization strings. In the same spirit as Section II-A, we show that synchronization strings can be used to guess the original position of symbols undergoing insertion-deletion errors via indexing. However, for synchronization strings, the repositioning can be done in an online fashion, i.e., the position of each symbol is guessed upon its arrival and without waiting for the rest of the communication to take place. This enables a delay-free simulation of a channel with Hamming-type errors over any given insertion-deletion channel with adequately large alphabet size. We will discuss this further in Section IV.

To present the online repositioning algorithm, we introduce the notion of relative suffix distance inspired by a similar notion from [50].

Definition III.3 (Relative Suffix Distance)**.** *For any $S, S' \in \Sigma^*$, their relative suffix distance (*RSD*) is defined as follows:*

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

It is shown in [39] that RSD is a metric that takes a value within $[0,1]$. The interesting property of RSD that comes in handy when devising an online repositioning algorithm is that the prefixes of a synchronization string are far apart under the RSD metric.

Proposition III.4. *Let S be an ε-synchronization string. For any $i \neq j$,* $\mathrm{RSD}(S[1,i], S[1,j]) > 1-\varepsilon$.

Note that an online repositioning algorithm is essentially one that decides which prefix of the message string is sent upon arrival of each symbol at the receiver side. Therefore, the online repositioning algorithm only needs to decide which prefix of the synchronization string is the most consistent to the index portion of the received string up until the arrival of each symbol. To this end, Proposition III.4 suggests the natural repositioning strategy of finding the closest prefix of the utilized synchronization string to the index part of the received string under relative suffix distance and declaring the length of that prefix as the position of that symbol.

The guarantees that this decoding strategy provides is discussed in details in [39]. However, we remark that the suffix distance between a string s and a noisy version of it, $\tilde{s}$, that is altered by insertions and deletions is particularly sensitive to how dense the fraction of error occurrences is in small suffixes of $\tilde{s}$. This implies that occurrences of insertions and deletions can only disrupt the correctness of this repositioning strategy for some of the following symbols and the effect would fade away as communication goes on. By formalizing these observations and employing a similar yet more complicated distance function, [39] gives the following.

Theorem III.5. *There exists an online repositioning algorithm for a communication of length n over a channel with up to $n\delta$ synchronization errors that, assuming that the message is indexed by an ε-synchronization string, guesses the position of each received symbol in $O(n^4)$ time and incorrectly guesses the positions of no more than $\frac{n\delta}{1-\varepsilon}$ received symbols.*

Note that, as opposed to the repositioning algorithm in Section II-A, the number of incorrect guesses does not tend to zero by taking smaller values for ε. In fact, if one constructs synchronization codes as in Section II-A with ε-synchronization strings and uses this repositioning algorithm instead of Algorithm 1, the rate achieved is $1-3\delta-\varepsilon^{O(1)}$.

C. Construction: Long-Distance Synchronization Strings

To construct synchronization strings, [40] utilizes the algorithmic Lovász local lemma of Chandrasekaran *et al*. [51] with a similar random space to the one used in Section III-A and obtains an efficient construction of such strings over an alphabet of size $O(\varepsilon^{-4})$. In this section, we review the steps taken in [40] to obtain a linear-time explicit construction for synchronization strings. In order to do so, we start with presenting the *long-distance synchronization string* property that generalizes the requirement of large edit distance to non-adjacent substrings that are at least logarithmically long in terms of the length of the string.

Definition III.6 (c-long-distance ε-synchronization string)**.** *String $S \in \Sigma^n$ is a c-long-distance ε-synchronization string if for every pair of substrings $S[i,j)$ and $S[i',j')$ that are either adjacent or of total length $c\log n$ or more, $ED\left(S[i,j), S[i',j')\right) > (1-\varepsilon)l$ where $l = j-i+j'-i'$.*

We now describe construction algorithms for (long-distance) synchronization strings.

1) Boosting Step I: Linear Time Construction: [40] provides a simple boosting step which allows a polynomial speed-up to any synchronization string construction at the cost of increasing the alphabet size by proposing a construction of an $O(\varepsilon)$-synchronization string of length $O_\varepsilon(n^2)$ having an ε-synchronization string of length n.

Lemma III.7. *Fix an even $n \in \mathbb{N}$ and $\gamma > 0$ such that $\gamma n \in \mathbb{N}$. Suppose $S \in \Sigma^n$ is an ε-synchronization string. The string $S' \in \Sigma'^{\gamma n^2}$ with $\Sigma' = \Sigma^3$ and*

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

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

Proof Sketch. S' is formed by the symbol-wise concatenation of three strings as presented in Eq. (2). The first two elements form repetitions of S which guarantee the synchronization property over small intervals and the third element that guarantees the synchronization property over larger intervals. □

Employing this boosting technique for an adequately large number of times can turn the polynomial-time construction of synchronization strings obtained by the algorithmic Lovász local lemma of [51] into a linear time construction at the cost of a larger alphabet that is still of $\varepsilon^{-O(1)}$ size.

2) Boosting Step II: Explicit Linear-Time Long-Distance Construction: We now describe a second boosting step introduced in [40] that takes the linear-time construction from the previous section and turns it into a linear-time construction for long-distance synchronization strings that is also highly-explicit, i.e., for any index i, it can compute the substring $[i, i+\log n]$ in $O(\log n)$ time.

To describe the construction, we first point out a connection between long-distance synchronization strings and synchronization codes. Note that if one splits a c-long-distance ε-synchronization string into substrings of length $c\log n$, the long-distance synchronization property will require that any pair of resulting substrings to have an edit distance of at least $2(1-\varepsilon)c\log n$, i.e., form an insertion-deletion code of relative distance $1-\varepsilon$.

Similarly, given a synchronization code C of distance $1-\varepsilon$, rate $r > 0$ and block length N, one can construct a string

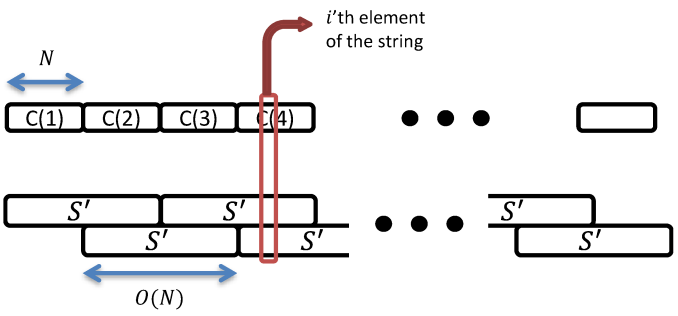

Fig. 2. Pictorial representation of the construction of a long-distance ε-synchronization string of length n.

of length $n = \exp(Nr)$ by appending the codewords of C together which satisfies the c-long-distance ε'-synchronization property for pairs of substrings of total length $\Omega(\log n)$ where $\varepsilon' = O(\varepsilon)$ and $c = O_\varepsilon(1)$. This claim is proved by a simple combinatorial argument using the distance property of the code C. We refer the reader to [40] for a formal proof.

The second boosting step uses this observation and makes a long-distance synchronization string by symbol-wise concatenation of the string described above with a string that guarantees the ε-synchronization property for neighboring intervals of total length $O(\log n)$. More formally, having the code C of block length N the construction is as follows.

$$S[i] = \left(C\left(\left\lfloor \frac{i}{N} \right\rfloor \right)[i \,(\mathrm{mod}\, N)], T[i] \right), \tag{3}$$

where T is the symbol-wise concatenation of two shifted repetitions of some synchronization string S' of length $O(\log n)$, i.e., $T[i] = (S'[i \bmod l], S'[(i + L/2) \bmod l])$ for $l = |S'|$. A pictorial description of the construction is available at Fig. 2. Given that the codewords of the code C take care of the long-distance synchronization property for longer pairs of intervals and repetitions of S' provide the synchronization string guarantee for short neighboring intervals, this construction yields a long-distance synchronization string. In order to show the linear-time and highly-explicit construction, the following two ingredients are necessary:

- A linear time construction for synchronization string S. This is provided by boosting step I.
- Linear time construction for code C. To obtain this, a family of high-rate synchronization codes with linear-time construction is necessary. To obtain such a family of codes, [40] takes the near-MDS code of Guruswami and Indyk [43] and concatenates it with an inner code to reduce its alphabet to $\varepsilon^{O(1)}$. Note that the size of the inner code is $O_\varepsilon(1)$. Therefore, the encoding time of the resulting family of codes remains linear and its rate is still positive. [40] then indexes the codewords of this code with an ε-self matching string as in Section II-A to obtain the necessary synchronization code for this construction.

The details of this construction are available in [40]. We summarize the guarantees of the construction in the following theorem.

Theorem III.8. *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 $|\Sigma| = \varepsilon^{-O(1)}$. This construction runs in linear time and, moreover, any substring $S[i, i + \log n]$ can be computed in $O_\varepsilon(\log n)$ time.*

D. Infinite Synchronization Strings

An infinite ε-synchronization string is naturally defined as an infinite string, in which, any two neighboring intervals $[i, j)$ and $[j, k)$ have an edit distance of at least $(1 - \varepsilon)(k - i)$. Existence of infinite ε-synchronization strings can be proved via a simple topological argument. Fix any $\varepsilon \in (0, 1)$. According to Theorem III.2 there exist an alphabet Σ of size $O(1/\varepsilon^2)$ such that there exists at least one ε-synchronization strings over Σ for every length $n \in \mathbb{N}$. We will define an infinite synchronization string $S = s_1 \cdot s_2 \cdot s_3 \cdots$ with $s_i \in \Sigma$ for any $i \in \mathbb{N}$ inductively. We fix an ordering on Σ and define $s_1 \in \Sigma$ to be the first symbol in this ordering such that an infinite number of these strings start with s_1. Given that there is an infinite number of ε-synchronization strings over Σ, such an s_1 exists. Furthermore, the subset of ε-synchronization strings over Σ which start with s_1 is infinite by definition, allowing us to define $s_2 \in \Sigma$ to be the lexicographically first symbol in Σ such there exists an infinite number of ε-synchronization strings over Σ starting with $s_1 \cdot s_2$. In the same manner, we inductively define the whole string. Since each prefix of this string satisfies the ε-synchronization property, all pairs of adjacent intervals satisfy the ε-synchronization property and this whole string is indeed an infinite ε-synchronization string.

The construction from Theorem III.8 can be generalized for infinite synchronization strings as follows.

Theorem III.9. *For all $0 < \varepsilon < 1$, there exists an infinite ε-synchronization string S over a $\mathsf{poly}(\varepsilon^{-1})$-sized alphabet so that any prefix of it can be computed in linear time. Further, for any i, $S[i, i + \log i]$ can be computed in $O(\log i)$ time.*

The proof of Theorem III.9 utilizes a construction for infinite synchronization strings obtained by concatenation of finite synchronization strings of exponentially increasing length. More precisely, let S_i denote a $\Theta(\varepsilon)$-synchronization string of length i. Further, let U and V be as follows:

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

Then [40] shows that the symbols-wise concatenation of these strings, i.e., string T where $T[i] = (U[i], V[i])$ is an infinite synchronization string. A pictorial representation of the construction of T is available in Fig. 3. The proof of Theorem III.9 is derived by simply using the above-mentioned construction and utilizing Theorem III.8 to construct the finite strings used to form U and V.

E. Local Decoding for Long-Distance Synchronization Strings

In this section, with a slight modification to the construction from (3) for long-distance synchronization strings, we give an index sequence which facilitates local repositioning. A *local*

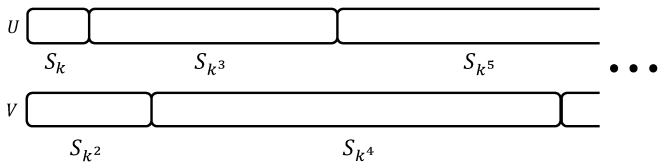

Fig. 3. Construction of infinite synchronization string T

repositioning algorithm is one that guesses the position of a received symbol using only the knowledge of a small $O(\log n)$-sized neighborhood of the surrounding received symbols, as opposed to all received symbols which is what Algorithm 1 does.

Consider the following indexing sequence that is obtained from the construction of (3) by concatenating an extra term that essentially works as a circular index counter for insertion-deletion code blocks.

$$R[i] = \left(C\left(\left\lfloor \frac{i}{N} \right\rfloor\right)[i \,(\mathrm{mod}\, N)], T[i], \left\lfloor \frac{i}{N} \right\rfloor \left(\mathrm{mod}\, \frac{8}{\varepsilon^3}\right)\right) \tag{4}$$

We claim that indexing the symbols of a communication over a synchronization channel with this string allows a repositioning algorithm which is both local and online.

Theorem III.10. *For a communication over a synchronization channel that is indexed by string R specified in* (4)*, there exists an online and local repositioning algorithm that guesses the position of each received symbol using only the symbol itself and $O_\varepsilon(\log n)$ symbols preceding it in $O_\varepsilon(\log^3 n)$ time. Also, among all symbols that are not deleted by the adversary, the position of no more than $\frac{n\delta}{1-\varepsilon}$ will be incorrectly guessed.*

We remark that, similar to Theorem III.5, the number of incorrect guesses does not tend to zero by taking smaller values for ε and synchronization codes constructed using such an index string would achieve a rate of $1 - 3\delta - \varepsilon^{O(1)}$.

Proof Sketch of Theorem III.10. The details of the decoding algorithm and the proof of its properties are presented in [40]. Here, we only give a high-level description of the algorithm and an informal justification of its correctness.

The analysis uses a notion called *suffix error density*, that is the maximum density of errors that have occurred over all suffixes of the communication up to that point. The algorithm guarantees to make a correct guess as long as the suffix error density is less than $1 - O(\varepsilon)$.

The decoding algorithm begins with using the first and the third elements of (4) (that are the codeword of $\mathcal{C}$ and the circular counter) to find the vicinity of the location; more precisely, which codeword of C the symbol belongs to or the quantity $\lfloor \frac{i}{N} \rfloor$. In order to do so, the algorithm looks at the $O\left(\frac{N}{\varepsilon^2}\right) = O\left(\frac{\log n}{\varepsilon^2}\right)$ last received symbols. If the value of the suffix error density is small, at least one of the previous $\frac{1}{\varepsilon}$ codewords of C appear inside this window with less than $1 - O(\varepsilon)$ synchronization errors. Let the value of the counter for the received symbol be c. With this observation, the repositioning algorithm runs the decoder of C over the subsequences of symbols with counter values $c, c-1, c-2, \cdots, c - \frac{1}{\varepsilon}$ in this window and comes up with a list of candidate vicinities for the position of the symbol. A second step uses the long-distance property of the string to choose one of these vicinities. Then, the exact position of the symbol within that vicinity is recovered using the repetitions of the small synchronization string that forms string T. (see (4)) □

F. Edit Distance Indexing and Near-linear Time Repositioning

As the final step in this section, we introduce a pseudo-random property for strings that, indexed to any given string, can facilitate edit distance computation. We then use such strings to enhance the construction of self-matching strings to obtain near-linear time repositioning algorithms.

1) Edit Distance Approximation via Indexing: In this section, we introduce an indexing scheme which can be used to approximate edit distance in near-linear time if one of the strings is indexed by an *edit-distance-approximating* string I. In particular, for every length n and every $\varepsilon > 0$, one can, in near-linear time, construct a string $I \in \Sigma'^n$ with $|\Sigma'| = O_\varepsilon(1)$, such that, indexing any string $S \in \Sigma^n$ with I results in the string $S \times I \in \Sigma''^n$ where $\Sigma'' = \Sigma \times \Sigma'$ and there is an algorithm that approximates the edit distance between $S \times I$ and any other string within a $1+\varepsilon$ factor and in $O(n \cdot \mathsf{poly}(\log n))$ time.

The construction of the index string I resembles the construction proposed for long-distance synchronization strings in Section III-C2. Namely, the index string I is simply constructed by writing, back-to-back, the codewords of a synchronization code that is list-decodable from high rates of synchronization errors, or more specifically, from any $1-\varepsilon'$ fraction of insertions and $1-\varepsilon'$ fraction of deletions for some $\varepsilon' = \Theta(\varepsilon)$. As we mentioned in Section II-C, it is shown in [24] that there exist families of codes that can be efficiently L-list-decoded from any $1-\varepsilon'$ fraction of insertions and $1-\varepsilon'$ fraction of deletions, achieve a rate of $\varepsilon'/2$, and are defined over an alphabet of size $\exp(\varepsilon'^{-3} \log \frac{1}{\varepsilon'}) = O_{\varepsilon'}(1)$. Here, L is some sub-polynomial function of the block length of the code.

Note that the properties of the code directly implies that the index string I is defined over an alphabet of size $O_\varepsilon(1)$ and can be computed in near-linear time. Let us take a member of the above-mentioned family of codes like C with block length N and denote its block length and its decoding function respectively with N and $\mathrm{Dec}_C(\cdot)$. Since the code has a positive rate, the length of the string formed by appending the codewords of C together would be $n = \exp(N)$ and therefore, the construction time for index string I is $\frac{n}{N} \cdot \mathsf{poly}(N) = O(n \cdot \mathsf{polylog}(n))$.

We now describe the algorithm that approximates the edit distance of S indexed with I ($S \times I$) to any given string S'. The algorithm starts by splitting the string S' into blocks of length N in the same spirit as $S \times I$. We denote the ith such block by $S'(i)$ and the ith block of $S \times I$ by $[S \times I](i)$. Note that the blocks of $S \times I$ are substrings of S indexed by the codewords of an insertion-deletion code with high distance ($[S \times I](i) = S[N(i-1), Ni - 1] \times C(i)$). Now, consider the set of insertions and deletions that correspond to the edit

distance between $S \times I$ and S' or n arbitrary one of them if there are more than one. One might expect that any block of S that is not significantly altered by such insertions and deletions, (i) appears in a set of consecutive blocks in S' and (ii) has a small edit distance to at least one of those blocks.

Following this intuition, our proposed algorithm works thusly: For any block of S' like $S'(i)$, the algorithm uses the list-decoder of $\mathcal{C}$ to find all (up to L) blocks of S that can be turned into $S'(i)$ by $N(1-\varepsilon)$ deletions and $N(1-\varepsilon)$ insertions only considering the index portion of the alphabet and ignoring the content portion of it. In other words, let $S'(i) = C'_i \times S'_0[N(i-1), Ni-1]$. We denote the set of such blocks by $\text{Dec}_{\mathcal{C}}(C'_i)$. Then, the algorithm constructs a bipartite graph G with $|S|$ and $|S'|$ vertices on each side (representing symbols of S and S') as follows: a symbol in $S'(i)$ is connected to all identical symbols in the blocks that appear in $\text{Dec}_{\mathcal{C}}(C'_i)$ or any block that is in their $w = O\left(\frac{1}{\varepsilon}\right)$ neighborhood, i.e., is up to $O\left(\frac{1}{\varepsilon}\right)$ blocks away from at least one of the members of $\text{Dec}_{\mathcal{C}}(C'_i)$.

Note that any non-crossing matching in G corresponds to some common subsequence between S and S' because G's edges only connect identical symbols. In the next step, the algorithm finds the largest non-crossing matching in G, $\mathcal{M}_{ALG}$, and outputs the corresponding set of insertions and deletions as the output. Finally, an algorithm proposed by Hunt and Szymanski [52] is used to compute the largest non-crossing matching of G with n vertices and r edges in $O\left((n+r)\log\log n\right)$. A formal description is available in Algorithm 2. As the number of edges of G cannot exceed $\frac{n}{\log n} \cdot \log^2 n = n \log n$ and code C is efficiently list-decodable, the run time for this algorithm is $O(n \cdot \text{polylog}(n))$.

Algorithm 2 $(1 + O(\varepsilon'))$-Approximation for Edit Distance

Input: $S \times I, S', N, \text{Dec}_{\mathcal{C}}(\cdot)$

1: Make empty bipartite graph $G(|S|, |S'|)$
2: $w = \frac{1}{\varepsilon'}$
3: **for each** $S'(i) = C'_i \times S'_0[N(i-1), Ni-1]$ **do**
4: $\quad List \leftarrow \text{Dec}_{\mathcal{C}}(C'_i)$
5: $\quad$ **for each** $j \in List$ **do**
6: $\quad\quad$ **for** $k \in [j-w, j+w]$ **do**
7: $\quad\quad\quad$ Connect pairs of vertices in G that correspond to identical symbols in $S(k)$ and $S'(i)$.
8: $\quad\quad$ **end for**
9: $\quad$ **end for**
10: **end for**
11: $\mathcal{M}_{ALG} \leftarrow$ Largest non-crossing matching in G ([52])

Output: $\mathcal{M}_{ALG}$

The detailed proof of the approximation guarantee is available in [42]. We provide a general proof sketch here.

Note that if graph G from Algorithm 2 contains the matching that corresponds to the LCS between $S \times I$ and S', then the algorithm will find the longest common subsequence in Line 11 and compute the exact edit distance. To show that Algorithm 2 finds a $1 + O(\varepsilon')$ approximation of the edit distance, [42] associates any edge from the LCS missing in G to $O(1/\varepsilon')$ insertions or deletions from the optimal edit distance solution.

Consider the matching that corresponds to the LCS. If some block of S' like $S'(i')$ is connected to more than $1 + \frac{1}{\varepsilon'}$ blocks in S, the unmatched vertices among those blocks account for $n \times \frac{1}{\varepsilon'}$ deletions in the optimal edit distance solution. Therefore, even if none of the edges of LCS that have an endpoint in such blocks appear in G, the size of the edit distance would increase by a factor of $1 + O(\varepsilon')$. This is why the parameter w is chosen as $\frac{1}{\varepsilon'}$ in Algorithm 2.

Further, if some block of S' is only connected to one block of S and has no more than $\frac{N}{\varepsilon}$ edges to it, $N - \frac{N}{\varepsilon}$ of its symbols are insertions in the optimal edit distance solution. Therefore, the absence of its edges from G in Algorithm 2 may only increase the size of the edit distance solution by a factor of $1 + O(\varepsilon')$.

In [42], the authors show that all LCS edges that are absent from G fall under these two categories and, therefore, the outcome of Algorithm 2 is an $1+O(\varepsilon') = 1+\varepsilon$ approximation.

2) Near-linear Time Repositioning: Note that the repositioning algorithm for strings indexed with ε-synchronization strings that was presented in Algorithm 1 consists of multiple rounds of edit distance computation between the synchronization string used and a distorted version of it. To reduce the run time of the repositioning algorithm, one can use the edit-distance approximating indexes from Section III-F1 and index ε-synchronizations strings with them. Then, use edit distance approximations instead of exact computations in Algorithm 1. We formally summarize this in the following.

Theorem III.11 (Theorem 7.1 of [42])**.** *Let S be a string of length n that consists of the symbol-wise concatenation of an ε_s-synchronization string and an edit distance indexing sequence from Section III-F1 with parameter ε_I. Assume that a stream of messages indexed by S goes through a channel that might impose up to $\delta \cdot n$ deletions and $\gamma \cdot n$ symbol insertions for some $0 \leq \delta < 1$ and $0 \leq \gamma$. For any positive integer K, there exists a repositioning algorithm that runs in $O(Kn \cdot \text{polylog}(n))$ time, guarantees up to $n\left(\frac{1+\gamma}{K(1+\varepsilon_I)} + \frac{\varepsilon_I(1+\gamma/2)}{1+\varepsilon_I} + K\varepsilon_s\right)$ incorrect guesses and does not decode more than K received symbols to any number in $[1, n]$.*

IV. Further Applications of Synchronization Strings

A. Codes for Block Transpositions and Replications

We showed in Section III-E that using long-distance synchronization strings in the indexing-based synchronization code construction allows local repositioning, i.e., the decoder will be able to guess the original position of each symbol by only looking at a logarithmically long neighborhood of the received symbol. In this section, we show that this property enables the code to protect from block transposition and block duplication errors as well.

Block transposition errors allow for arbitrarily long substrings of the message to be moved to another position in the message string. Similarly, block duplication errors are ones that pick a substring of the message and copy it between two symbols of the communication.

We will present codes that can achieve a rate of $1-\delta-\varepsilon$ and correct from some $O(\delta)$ fraction of synchronization errors, a $O(\delta/\log n)$ fraction of block errors, or a combination of them. A similar result for insertions, deletions, and block transpositions was shown by Schulman and Zuckerman [18] where they provided the first constant-distance and constant-rate synchronization code correcting from insertions, deletions, and block errors. They also show that the $O(\delta/\log n)$ resilience against block errors is optimal up to constants.

Theorem IV.1. *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.*

Proof Sketch. Similar to Section II-A, this code is constructed by indexing near-MDS codes of Guruswami and Indyk [43] with a pseudo-random string, particularly, long-distance synchronization strings. The decoding procedure also follows the same steps as Section II-A. Namely, the decoder uses the repositioning algorithm presented in Theorem III.10 to guess the actual position of the symbols and then runs the decoder of the Guruswami-Indyk code over the reconstructed string.

Note that the repositioning guarantee from Theorem III.10 implies that with the choice of some small ε parameter for the long-distance synchronization string, the repositioning algorithm correctly guesses the position of all but $O(n\delta_{insdel})$ symbols where n is the length of the communication if only insertions and deletions are allowed.

Additionally, the local quality of the repositioning algorithm implies that any symbol at the receiver that does not have any synchronization errors or block error borders in its $O(\log n)$ neighborhood, is correctly repositioned by the local repositioning algorithm. Therefore, with $n\delta_{block}$ block errors, no more than $n\delta_{block}\log n$ repositioning guesses would be incorrect. This implies an $O(n/\log n)$ block error resilience. Combining the two remarks above gives that the code can correct $n\delta_{insdel}$ 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 constant c.

The encoding and decoding complexities simply follow the properties of the Guruswami-Indyk codes, linear time constructions of long-distance synchronization strings from Theorem III.8 and time complexity of the repositioning algorithm from Theorem III.10. □

B. Channel Simulation

The construction of codes based on indexing presented in this paper suggests that indexing with pseudo-random strings can reduce synchronization errors to more benign Hamming-type errors (substitutions and erasures). In this section, we present results from [21], [40] which shows that this is indeed true.

More precisely, having a channel afflicted by synchronization errors, one can put two simulation agents on the two ends of the channel who can simulate a channel with Hamming-type errors over the given channel. In other words, the sender/receiver sends/receives symbols to/from their corresponding agent and the simulation guarantees that the channel would seem like a channel with Hamming-type errors to the parties.

Note that the indexing scheme from Section II-A almost achieves this goal by reducing synchronization errors to half-errors through indexing. However, this procedure requires all symbols to be communicated before the repositioning procedure of Algorithm 1 can start running and, therefore, introduces a delay. A true channel simulation would not add such delay. More precisely, a round of error-free communication in a simulated channel is one that communicates the ith symbol sent by the sender as the ith symbol to the receiver once it arrives at the other side and prior to the $i+1$st symbol being sent by the sender.

This subtle requirement can be satisfied through using synchronization strings as the indexing sequence and utilizing the online repositioning algorithm introduced in Section III-B.

Before presenting the channel simulations, we remark an interesting negative result of [21] stating that, as opposed to codes, when it comes to channel simulations, no channel simulator can reduce δ fraction of synchronization errors to $\delta+\varepsilon$ half-errors for arbitrarily small ε.

Theorem IV.2. *Assume that n uses of a synchronization channel over an arbitrarily large alphabet Σ with a δ fraction of insertions and deletions are given. There is no deterministic simulation of a half-error channel over any alphabet Σ_{sim} where the simulated channel guarantees more than $n(1-4\delta/3)$ uncorrupted transmitted symbols. If the simulation is randomized, the expected number of uncorrupted transmitted symbols is at most $n(1-7\delta/6)$.*

We now present the channel simulations that can be achieved via indexing with synchronization strings. Simulations are presented for channels with large constant alphabets, binary alphabets, one-way communication or interactive communication.

Theorem IV.3. *[Channel Simulations]*

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

(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 substitution channel with $\Theta(\sqrt{\delta\log(1/\delta)})$ fraction of substitution errors between two parties over the given channel.*

All of the simulations mentioned above take $O(1)$ time per symbol for the 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 $O(\log n)$ 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 Sketch. We highlight the main ideas behind each of these simulations in the following.

(a) In this simulation, the simulating agents do the indexing as done in the case of coding. Meaning that on the sender side, the simulation simply indexes the messages of the sender with symbols of a long-distance synchronization string and on the receiving end, the receiver-side simulating agent runs the online repositioning algorithm from Section III-E to identify the position of the symbols it receives and relays them to the receiver.
Note that the online repositioning algorithm for long-distance synchronization strings allows the simulator on the receiving end to guess the positions of the received symbols as they arrive. However, we stress that the simulated channel has to behave as an actual channel and therefore cannot reveal the symbols to the receiver out of order. For instance, if the repositioning algorithm incorrectly identifies the first symbol as the tenth symbol and reveals it as the tenth symbol to the receiver, it cannot reveal the second to ninth symbols to the receiver afterwards even if the repositioning for those symbols is done correctly.
To ensure in-order revealing of symbols, the simulation uses a lazy revealing strategy to avoid over-reacting to incorrect guesses by the repositioning algorithm. More precisely, if the guessed position of a symbol is far beyond where the communication length is at that moment, the receiver-side simulator moves the communication forward by outputing two dummy symbols to the reciever. For the analysis of this strategy, we refer the reader to [21].

(b) The simulation for channels with a binary alphabet is very similar except that the indexing is not possible due to the size of the alphabet. To overcome this, the simulation splits the communication into several blocks. In each block, the sender-side simulator first sends a fixed header of size $O(\log \frac{1}{\delta})$ (indicating the start of a new block), then sends a binary encoding of a symbol of the long-distance synchronization string, and then ends the block by relaying the messages of the sender for $r = \sqrt{\frac{\log 1/\delta}{\delta}}$ rounds.

Time and space guarantees of these simulations are inferred from highly-explicit constructions of infinite long-distance synchronization strings (Theorem III.9) and their local repositioning algorithms (Theorem III.10). Similar simulations can be performed in interactive communication channels by taking the steps mentioned above in one direction of the communication. □

C. Interactive Communication for Synchronization Errors

The channel simulations via indexing presented in Section IV-B can be used to obtain interactive coding schemes for synchronization errors. Interactive communication between two parties is one in which any round of the communication consists of a message transmission from one party to the other one. Each party is assumed to hold a private information denoted by X and Y and the goal is for both parties to compute some function $f(X, Y)$. Any strategy for computing $f(X, Y)$ is called a protocol.

A coding scheme for interactive communication is one that takes any protocol that computes some function f in noiseless communication and converts it into a protocol that computes f over a noisy channel. The rate of an interactive coding scheme is defined as the minimal ratio of the length of the protocol in the absence of noise over the length of the protocol in the presence of noise over all functions f.

The channel model used in the results of this section is the commonly used model of Braverman *et al*. [50] that considers an alternating protocol, i.e., protocols in which parties take alternating turns in sending and receiving symbols.

Using channel simulations, [21], [40] provide coding schemes for interactive communication over channels suffering from synchronization errors by simply simulating a half-error channel over the given synchronization channel and applying interactive protocols for channels with symbol substitution errors over the simulated channel. Using simulations for channels with large alphabets along with the interactive protocol of Haeupler and Ghaffari [53], [40] gives the following.

Theorem IV.4. *For a sufficiently small δ and n-round alternating protocol Π, there is a randomized coding scheme simulating Π in the presence of δ fraction of synchronization errors with constant rate (i.e., in $O(n)$ rounds) and in near-linear time. This coding scheme works with probability $1 - 2^{\Theta(n)}$.*

Similarly, using binary alphabet simulations and the interactive protocol of Haeupler [54], [21] gives the following.

Theorem IV.5. *For sufficiently small δ, there is an efficient interactive coding scheme for fully adversarial binary synchronization channels which is robust against δ fraction of edit-corruptions, achieves a communication rate of $1 - \Theta(\sqrt{\delta \log(1/\delta)})$, and works with probability $1 - 2^{-\Theta(n\delta)}$.*

D. Binary Synchronization Codes

A similar approach is taken to design binary synchronization error-correcting codes in [21] by simulating a half-error channel over the given synchornization channel and then using a binary error-correcting code on top of it.

Theorem IV.6. *For any sufficiently small δ, there is a binary synchronization code with rate $1 - \Theta\left(\sqrt{\delta \log \frac{1}{\delta}}\right)$ which is decodable from δ fraction of insertions and deletions.*

It is shown in [10] that the optimal rate for binary synchronization codes with distance δ is $1 - O\left(\delta \log \frac{1}{\delta}\right)$. Recent works by Cheng *et al*. [22] and Haeupler [23] have simultaneously

improved over the codes from Theorem IV.6 by introducing efficient binary codes with rate $1 - O(\delta \log^2 \frac{1}{\delta})$ via providing deterministic document exchange protocols.

E. Document Exchange

Document exchange is a problem in which a server and a client hold two versions of the same string, say F and F' respectively, where F' is an outdated version that is different from F by up to k insertions or deletions. The goal is for the server to compute a small summary and send it to the client so the client can update its string to F.

There is a close connection between deterministic document exchange protocols and systematic synchronization codes. Having a systematic synchronization code, one can construct a document exchange protocol with using the non-systematic part of the code as the summary. On the other hand, having a document exchange protocol, one can construct a systematic code by taking the summary of the document exchange protocol, encoding it using a synchronization code, and using the encoded summary as the non-systematic part of the code.

For document exchange with k errors, $\Omega(k \log \frac{n}{k})$ bits of information is necessary as the summary. Orlitsky [55] showed in 1991 that protocols with this amount of redundancy exist, however, fell short of providing efficient ones. In 2005, Irmak, Mihaylov and Suel [56] provided an efficient document exchange protocol with $O(k \log \frac{n}{k} \log n)$ redundancy. Since then, there have been several works on randomized document exchange protocols (mostly for k sublinear in n) by [57], [58], [59], [60]. Recent works of Cheng *et al*. [22] and Haeupler [23] provide deterministic document exchange protocols with redundancy $O(k \log^2 \frac{n}{k})$.

Haeupler [23] first provides a randomized document exchange protocol with redundancy $O(\delta \log \frac{1}{\delta})$ through a modification and careful analysis of the protocol of Irmak *et al*. [56]. Then, it derandomizes the protocol using a derandomizarion technique reminiscent of one used in [40] of synchronization strings. The techniques of Cheng *et al*. [22] also make use of notions called ε-self-matching hash functions and ε-synchronization hash functions which are sequences of hash functions whose outputs satisfy properties resembling the corresponding string properties introduced in [39].

F. Linear Insertion-Deletion Codes

A recent work of Cheng *et al.* [61] studies qualities like linearity and affinity in the context of synchronization coding. They propose an efficient synchronization string-based transformation that can convert any asymptotically good linear error-correcting code into an asymptotically good insertion-deletion code. Using this transformation along with well-known linear error-correcting codes, such as Hamming codes, results in explicit constructions for linear insertion-deletion codes.

The indexing scheme introduced in Section II inherently leads to codes that are non-linear as it specifies a fixed value to a portion of each symbol. To circumvent this, [61] uses pseudo-random strings to design linear insertion-deletion codes in the following manner: To encode a message $x \in \mathbb{F}_q^m$, they first encode it using a linear error-correcting code for Hamming-type errors $C : \mathbb{F}_q^m \to \mathbb{F}_q^n$ to become $y = C(x)$. They then take care of the synchronization issues by inserting several sequences of 0 symbols into the message and generating the final codeword as follows:

$$z_? = (0^{S_1}, y_1, 0^{S_2}, y_2, \cdots, 0^{S_n}, y_n).$$

The string $S = (S_1, S_2, \cdots, S_n)$ is a pseudo-random string having synchronization properties similar to the ones studied in this survey.

More precisely, [61] defines a Λ-*synchronization separator sequence* as a sequence S for which any $z = (0^{S_1}, ?, 0^{S_2}, ?, \cdots, 0^{S_n}, ?)$ does not have self-matchings with more than Λ *undesirable* matches. An undesirable match is one between two '?'s like the ith and the jth '?' where $i \neq j$ and $p_i - p_{i'} = p_j - p_{j'}$ where (i', j') is the immediate previous match to (i, j) in the matching and p_i denote the position of the ith '?' in $z_?$.

[61] provides explicit constructions for synchronization separator sequences and shows that, if used in the above construction, they enable the decoder to reconstruct the codeword y up to a number of Hamming-type errors that is within a constant factor of the number of insertions and deletions applied; Hence, proving that this conversion preserves both linearity and the asymptotic goodness of the code.

G. Coded Trace Reconstruction

A recent work by Brakensiek *et al.* [62] provides novel results for the coded trace reconstruction problem. Coded trace reconstruction asks for codes that satisfy the following: Assuming that a sender chooses a codeword of the code and sends multiple copies of it over independent binary deletion channels (called *traces*), the receiver wants to be able to recover the original codeword with high probability. Using synchronization strings, [62] provides a high-rate coded trace reconstruction scheme that is efficiently decodable from a constant number of traces.

H. Coding for Binary Deletion Channels and Poisson Repeat Channels

Con and Shpilka [63] use the synchronization string-based Singleton-bound-approaching synchronization codes from Section I-C1 to provide an efficient and explicit code for binary deletion channels that improve over the state-of-the-art in terms of error resilience. They, additionally, show that their code also works for the Poisson repeat channel where each bit appears on the receiver's side a number of times which follows some Poisson distribution.

REFERENCES

[1] S. Golomb, J. Davey, I. Reed, H. Van Trees, and J. Stiffler, "Synchronization," *IEEE Transactions on Communications Systems*, vol. 11, no. 4, pp. 481–491, 1963.

[2] H. Mercier, V. K. Bhargava, and V. Tarokh, "A survey of error-correcting codes for channels with symbol synchronization errors," *IEEE Communications Surveys & Tutorials*, vol. 12, no. 1, pp. 87–96, 2010.

[3] M. Mitzenmacher, "A survey of results for deletion channels and related synchronization channels," *Probability Surveys*, vol. 6, pp. 1–33, 2009.

[4] L. Organick, S. D. Ang, Y.-J. Chen, R. Lopez, S. Yekhanin, K. Makarychev *et al.*, "Random access in large-scale DNA data storage," *Nature biotechnology*, vol. 36, no. 3, pp. 242–248, 2018.

[5] M. Blawat, K. Gaedke, I. Huetter, X.-M. Chen, B. Turczyk, S. Inverso *et al.*, "Forward error correction for DNA data storage," *Procedia Computer Science*, vol. 80, pp. 1011–1022, 2016.

[6] N. Goldman, P. Bertone, S. Chen, C. Dessimoz, E. M. LeProust, B. Sipos, and E. Birney, "Towards practical, high-capacity, low-maintenance information storage in synthesized DNA," *Nature*, vol. 494, no. 7435, p. 77, 2013.

[7] G. M. Church, Y. Gao, and S. Kosuri, "Next-generation digital information storage in DNA," *Science*, vol. 337, no. 6102, pp. 1628–1628, 2012.

[8] S. H. T. Yazdi, H. M. Kiah, E. Garcia-Ruiz, J. Ma, H. Zhao, and O. Milenkovic, "DNA-based storage: Trends and methods," *IEEE Transactions on Molecular, Biological and Multi-Scale Communications*, vol. 1, no. 3, pp. 230–248, 2015.

[9] J. Bornholt, R. Lopez, D. M. Carmean, L. Ceze, G. Seelig, and K. Strauss, "A DNA-based archival storage system," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 2, pp. 637–649, 2016.

[10] V. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Doklady Akademii Nauk SSSR*, vol. 163, no. 4, pp. 845–848, 1965, English translation in *Soviet Physics Doklady*, 10(8):707–710, 1966.

[11] M. Cheraghchi and J. Ribeiro, "An overview of capacity results for synchronization channels," *IEEE Transactions on Information Theory*, 2020.

[12] N. J. Sloane, "On single-deletion-correcting codes," *Codes and Designs*, vol. 10, pp. 273–291, 2002.

[13] G. Tenengolts, "Nonbinary codes, correcting single deletion or insertion (corresp.)," *IEEE Transactions on Information Theory*, vol. 30, no. 5, pp. 766–769, 1984.

[14] A. S. J. Helberg and H. C. Ferreira, "On multiple insertion/deletion correcting codes," *IEEE Transactions on Information Theory*, vol. 48, no. 1, pp. 305–308, 2002.

[15] R. Gabrys and F. Sala, "Codes correcting two deletions," *IEEE Transactions on Information Theory*, vol. 65, no. 2, pp. 965–974, 2018.

[16] K. A. S. Abdel-Ghaffar, F. Paluncic, H. C. Ferreira, and W. A. Clarke, "On Helberg's generalization of the Levenshtein code for multiple deletion/insertion error correction," *IEEE Transactions on Information Theory*, vol. 58, no. 3, pp. 1804–1808, 2011.

[17] J. Brakensiek, V. Guruswami, and S. Zbarsky, "Efficient low-redundancy codes for correcting multiple deletions," *IEEE Transactions on Information Theory*, vol. 64, no. 5, pp. 3403–3410, 2017.

[18] L. J. Schulman and D. Zuckerman, "Asymptotically good codes correcting insertions, deletions, and transpositions," *IEEE Transactions on Information Theory*, vol. 45, no. 7, pp. 2552–2557, 1999.

[19] V. Guruswami and R. Li, "Efficiently decodable insertion/deletion codes for high-noise and high-rate regimes," in *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, 2016, pp. 620–624.

[20] V. Guruswami and C. Wang, "Deletion codes in the high-noise and high-rate regimes," *IEEE Transactions on Information Theory*, vol. 63, no. 4, pp. 1961–1970, 2017.

[21] B. Haeupler, A. Shahrasbi, and E. Vitercik, "Synchronization strings: Channel simulations and interactive coding for insertions and deletions," in *Proceedings of the International Conference on Automata, Languages, and Programming (ICALP)*, 2018, pp. 75:1–75:14.

[22] K. Cheng, Z. Jin, X. Li, and K. Wu, "Deterministic document exchange protocols, and almost optimal binary codes for edit errors," in *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 2018.

[23] B. Haeupler, "Optimal document exchange and new codes for insertions and deletions," in *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 2019, pp. 334–347.

[24] B. Haeupler, A. Shahrasbi, and M. Sudan, "Synchronization strings: List decoding for insertions and deletions," in *Proceedings of the International Conference on Automata, Languages, and Programming (ICALP)*, 2018.

[25] B. Haeupler and A. Shahrasbi, "Rate-distance tradeoffs for list-decodable insertion-deletion codes," *arXiv preprint arXiv:2009.13307*, 2020.

[26] A. Wachter-Zeh, "List decoding of insertions and deletions," *IEEE Transactions on Information Theory*, vol. 64, no. 9, pp. 6297–6304, 2018.

[27] T. Hayashi and K. Yasunaga, "On the list decodability of insertions and deletions," *IEEE Transactions on Information Theory*, vol. 66, no. 9, pp. 5335–5343, 2020.

[28] B. Bukh, V. Guruswami, and J. Håstad, "An improved bound on the fraction of correctable deletions," *IEEE Transactions on Information Theory*, vol. 63, no. 1, pp. 93–103, 2017.

[29] S. Liu, I. Tjuawinata, and C. Xing, "List decoding of insertion and deletion codes," *arXiv preprint arXiv:1906.09705*, 2019.

[30] V. Guruswami, B. Haeupler, and A. Shahrasbi, "Optimally resilient codes for list-decoding from insertions and deletions," in *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 2020, pp. 524–537.

[31] F. Sellers, "Bit loss and gain correction code," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 35–38, 1962.

[32] H. Morita, A. Van Wijngaarden, and A. H. Vinck, "Prefix synchronized codes capable of correcting single insertion/deletion errors," in *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, 1997, p. 409.

[33] W. Ferreira, W. A. Clarke, A. S. J. Helberg, K. A. S. Abdel-Ghaffar, and A. H. Vinck, "Insertion/deletion correction with spectral nulls," *IEEE Transactions on Information Theory*, vol. 43, no. 2, pp. 722–732, 1997.

[34] E. Gilbert, "Synchronization of binary messages," *IRE Transactions on Information Theory*, vol. 6, no. 4, pp. 470–477, 1960.

[35] L. J. Guibas and A. M. Odlyzko, "Maximal prefix-synchronized codes," *SIAM Journal on Applied Mathematics*, vol. 35, no. 2, pp. 401–418, 1978.

[36] A. Van Wijngaarden and B. Morita, "Extended prefix synchronization codes," in *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, 1995, p. 465.

[37] H. Morita, A. J. van Wijngaarden, and A. H. Vinck, "On the construction of maximal prefix-synchronized codes," *IEEE Transactions on Information Theory*, vol. 42, no. 6, pp. 2158–2166, 1996.

[38] W. Kautz, "Fibonacci codes for synchronization control," *IEEE Transactions on Information Theory*, vol. 11, no. 2, pp. 284–292, 1965.

[39] B. Haeupler and A. Shahrasbi, "Synchronization strings: Codes for insertions and deletions approaching the Singleton bound," in *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 2017, pp. 33–46.

[40] ——, "Synchronization strings: Explicit constructions, local decoding, and applications," in *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 2018, pp. 841–854.

[41] A. Rubinstein, "Approximating edit distance," https://theorydish.blog/2018/07/20/approximating-edit-distance/, 2018.

[42] B. Haeupler, A. Rubinstein, and A. Shahrasbi, "Near-linear time insertion-deletion codes and $(1+\varepsilon)$-approximating edit distance via indexing," in *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 2019, pp. 697–708.

[43] V. Guruswami and P. Indyk, "Linear-time encodable/decodable codes with near-optimal rate," *IEEE Transactions on Information Theory*, vol. 51, no. 10, pp. 3393–3400, 2005.

[44] B. Hemenway, N. Ron-Zewi, and M. Wootters, "Local list recovery of high-rate tensor codes and applications," *SIAM Journal on Computing*, vol. 49, no. 4, pp. FOCS17–157–FOCS17–195, 2020.

[45] S. Kopparty, N. Resch, N. Ron-Zewi, S. Saraf, and S. Silas, "On list recovery of high-rate tensor codes," *IEEE Transactions on Information Theory*, 2020.

[46] B. Bukh and J. Ma, "Longest common subsequences in sets of words," *SIAM Journal on Discrete Mathematics*, vol. 28, no. 4, pp. 2042–2049, 2014.

[47] E. Arikan, "Channel polarization: a method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on Information Theory*, vol. 55, no. 7, pp. 3051–3073, 2009.

[48] J. Blasiok, V. Guruswami, P. Nakkiran, A. Rudra, and M. Sudan, "General strong polarization," in *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 2018, pp. 485–492.

[49] K. Cheng, B. Haeupler, X. Li, A. Shahrasbi, and K. Wu, "Synchronization strings: highly efficient deterministic constructions over small alphabets," in *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2019, pp. 2185–2204.

[50] M. Braverman, R. Gelles, J. Mao, and R. Ostrovsky, "Coding for interactive communication correcting insertions and deletions," *IEEE Transactions on Information Theory*, vol. 63, no. 10, pp. 6256–6270, 2017.

[51] K. Chandrasekaran, N. Goyal, and B. Haeupler, "Deterministic algorithms for the lovász local lemma," *SIAM Journal on Computing*, vol. 42, no. 6, pp. 2132–2155, 2013.

[52] J. W. Hunt and T. G. Szymanski, "A fast algorithm for computing longest common subsequences," *Communications of the ACM*, vol. 20, no. 5, pp. 350–353, 1977.
[53] M. Ghaffari and B. Haeupler, "Optimal error rates for interactive coding II: Efficiency and list decoding," in *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 2014, pp. 394–403.
[54] B. Haeupler, "Interactive channel capacity revisited," in *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 2014, pp. 226–235.
[55] A. Orlitsky, "Interactive communication: Balanced distributions, correlated files, and average-case complexity," in *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 1991, pp. 228–238.
[56] U. Irmak, S. Mihaylov, and T. Suel, "Improved single-round protocols for remote file synchronization," in *Proceedings of the Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3, 2005, pp. 1665–1676.
[57] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith, "Fuzzy extractors: How to generate strong keys from biometrics and other noisy data," *SIAM Journal on Computing*, vol. 38, no. 1, pp. 97–139, 2008.
[58] H. Jowhari, "Efficient communication protocols for deciding edit distance," in *European Symposium on Algorithms*. Springer, 2012, pp. 648–658.
[59] D. Chakraborty, E. Goldenberg, and M. Koucký, "Streaming algorithms for embedding and computing edit distance in the low distance regime," in *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 2016, pp. 712–725.
[60] D. Belazzougui and Q. Zhang, "Edit distance: sketching, streaming, and document exchange," in *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 2016, pp. 51–60.
[61] K. Cheng, V. Guruswami, B. Haeupler, and X. Li, "Efficient linear and affine codes for correcting insertions/deletions," in *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2020.
[62] J. Brakensiek, R. Li, and B. Spang, "Coded trace reconstruction in a constant number of traces," in *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 2020.
[63] R. Con and A. Shpilka, "Explicit and efficient constructions of coding schemes for the binary deletion channel," in *Proceedings of the IEEE International Symposium on Information Theory (ISIT)*, 2020, pp. 84–89.

Bernhard Haeupler is an Associate Professor of Computer Science at Carnegie Mellon University. He received his Ph.D. and M.Sc. in Computer Science from the Massachusetts Institute of Technology (MIT), and a B.Sc., M.Sc. and Diploma in (Applied) Mathematics from the Technical University of Munich. He spent a year at Princeton University and a year as a research scientist at Microsoft Research Silicon Valley. Haeupler has (co-)authored over 100 publications and patents. His research won many awards, including a George Sprowls Award for an outstanding PhD thesis in Computer Science and Electrical Engineering at MIT, the 2014 ACM-EATCS Doctoral Dissertation Award of Distributed Computing, three best (student) paper awards, an NSF Faculty Early Career Development award (NSF CAREER), a Sloan Research Fellowship, and an ERC Starting Grant. His research interests lie in the intersection of design and analysis of randomized and combinatorial algorithms, distributed computing, and (network) coding theory.

Amirbehshad Shahrasbi is an incoming Postdoctoral Fellow at Harvard University. He is a recipient of the NSF-funded Computing Innovation Fellowship (CIFellowship 2020) from the Computing Research Association (CRA). He received his Ph.D. in Computer Science from Carnegie Mellon University in 2020. During the summer of 2019, he worked as an intern at Microsoft Research's DNA Data Storage group. Previously, he had received a B.Sc. in Computer Science and a B.Sc. in Electrical Engineering from Sharif University of Technology in 2015. His research is currently focused on algorithmic coding theory.