DETERMINISTIC CONSTRUCTION OF
SYNCHRONIZATION STRING OVER SMALL ALPHABET

by

Ke Wu

A thesis submitted to Johns Hopkins University in conformity with the requirements for the degree of Master of Science in Engineering.

Baltimore, Maryland

December, 2017
Abstract

Synchronization string, first introduced by Haeupler and Shahrasbi [10], is a strong tool in construction of error correcting codes for insertion and deletion errors (insdel codes). Synchronization string provides a way to encode the indices of symbols in a string and makes it possible to transfer synchronization errors to easier half errors, which is much better understood.

In this paper, we improve the construction in [10] in the following aspects:

- We achieve a \textbf{smaller alphabet size}, reduce the alphabet size from $O(\varepsilon^{-4})$ in [10] to $O(\varepsilon^{-2})$.

- We give an \textbf{efficient deterministic construction of synchronization string} over alphabet of size $O(\varepsilon^{-3})$.

- We give a \textbf{near linear deterministic construction of synchronization string} over alphabet of size $O(\varepsilon^{-4})$. This algorithm runs in $O(n \log^2 \log n)$. Independently, Haeupler and Shahrasbi give a linear, deterministic construction for synchronization string over alphabet of $\varepsilon^{-O(1)}$ in their work [11].

- We introduce a combinatorial object called synchronization circle which enhances the property of synchronization string.

\textbf{Advisor: Xin Li}
Acknowledgement

I would like to thank sincerely Professor Xin Li and Kuan Cheng for their significant influence on this thesis.

I’d also like to appreciate Yanif Ahmad, Vladimir Braverman, Randal Burns, Zachary Burwell, Michael Dinitz, Mark Dredze, Abhishek Jain, Philipp Koehn and Revelie Niles for their great help in my graduate study.

Finally, I would like to thank my beloved Bingkai for his encouragement and companionship.
## Contents

Abstract .......................................................... ii

Acknowledgement ................................................... iii

List of Figures ......................................................... vi

1 Introduction ......................................................... 1
   1.1 Our Result ......................................................... 2
      1.1.1 Existence of Synchronization String over Alphabet of Size $O(\epsilon^{-2})$ ........................................ 2
      1.1.2 Near Linear Deterministic Construction of Synchronization String ........................................ 3

2 Definitions and Preliminaries .................................... 4
   2.1 Notations ......................................................... 4
   2.2 Definitions ....................................................... 5

3 $\epsilon$-synchronization Strings and Circles with Alphabet Size $O(\epsilon^{-2})$ .................. 6
   3.1 Existence and Random Construction of Synchronization String 6
   3.2 Synchronization circle .......................................... 12

4 Deterministic Constructions ...................................... 18

Reference ............................................................. 31
Biography
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Example where $S'[i, k]$ contains $s_n, s_1$ but doesn’t contain $s_{[n/2]}$</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>Example where $S'[i, k]$ contains $s_{[n/2]}, s_{[n]}$</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>Example where $S'[i, k]$ contains $s_{[n/2]}$ and $s_n$</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>Example of match between $S_1$ and $S_2$</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>Example in which $k - i &gt; m, m \mid j$ and $\ell_1 + \ell_2 = 0$</td>
<td>21</td>
</tr>
<tr>
<td>6</td>
<td>Example in which $k - i &gt; m, m \mid j, \ell_1 = 0, \ell_2 = 1$</td>
<td>22</td>
</tr>
<tr>
<td>7</td>
<td>Example in which $k - i &gt; m, m \nmid j, \ell_1 + \ell_2 = 0$</td>
<td>22</td>
</tr>
<tr>
<td>8</td>
<td>Example in which there’s match between $c_1^0$ and $c_2^0$</td>
<td>23</td>
</tr>
<tr>
<td>9</td>
<td>Example in which there’s no match between $c_1^0$ and $c_2^0$</td>
<td>24</td>
</tr>
</tbody>
</table>
1 Introduction

The study over insertion and deletion errors, which are called synchronization errors, are lagged far behind half errors which includes erasures and corruptions. Intuitively, synchronization errors is more general and difficult to handle than half errors because we don’t know the original index even for an unmodified symbol in the original words. In the contrary, in half error settings the position of each symbol stays the same. Naturally a way to encode the indices will help to transform synchronization errors to half errors.

The naive indices $1, 2, 3, \ldots, n$ is not what we desired as it has an alphabet size depending on the length of the string. In a recent work [10], Haeupler and Shahrasbi introduced a strong object synchronization string which provides a novel way of encoding the indices of symbols in a string.

**Definition 1.1** ($\varepsilon$-synchronization string). A string $S$ is an $\varepsilon$-synchronization string if $\forall 1 \leq i \leq j \leq k \leq n + 1$, 

$$ED(S[i, j], S[j, k]) \geq (1 - \varepsilon)(k - i)$$

Synchronization string, which is non-self-similar sequences, allows to finish the encoding with a finite alphabet size of $\varepsilon^{-O(1)}$ for arbitrary length. Along with the appropriate decoding algorithm, one can get the correct indices of these symbols and can thus use a standard error correcting code to recover the original word.

Besides constructing insdel code (insertion and deletion code), synchro-
nization string has a wide application like synchronization sequences [14], interactive coding schemes [4, 5, 7, 6, 8, 13, 12], and edit distance tree codes [3].

1.1 Our Result

Here we give an overview over the results and structure of this paper.

1.1.1 Existence of Synchronization String over Alphabet of Size $O(\varepsilon^{-2})$

In [10] the authors utilized Lovsz Local Lemma to show that a random string can be a synchronization string with a positive probability. Considering that synchronization string is non-self-similar, namely that, the symbols in the string should be as distinct as possible, we use a non-uniform sample space which largely enhances the diversity in the symbols than uniform sample space used in [10]. Specifically, we fix an alphabet $\Sigma$ of size $O(\varepsilon^{-2})$ and a number $t$ smaller than the size of $\Sigma$. We sample the symbols as follows: uniformly randomly pick first $t$ symbols from $\Sigma$ without replacement. Then for later positions, we uniformly randomly choose symbols different from previous $t - 1$ symbols from $\Sigma$. This way of choosing symbols guarantees that there's no duplicate in every interval of length $t$. With a similar proof in [10] we show that longer intervals are also non-self-similar with a positive probability using Lovsz Local Lemma. And we have the following results.
Theorem 1.2. There exists an $\varepsilon$-synchronization string $S$ of length $n$ for $\varepsilon \in (0, 1)$ and $n \geq 1$ over alphabet $\Sigma$ of size $O(\varepsilon^{-2})$.

Similarly with [10], this existence proof together with algorithmic Lovsz Local Lemma in [16] and the extension in [9] give a random algorithm constructing synchronization string.

1.1.2 Near Linear Deterministic Construction of Synchronization String

A synchronization string $S$ is non-self-similar, which means that the edit distance of non-intersect intervals should be as large as possible. Note that if two substring have same length, then they can be regarded as two codewords for an insdel code due to their large distance. This fact suggests that one can use codewords of an insdel code to construct a synchronization string. With carefully chosen parameters, we show that a synchronization string can be constructed through concatenation of different codewords of an insdel code. However, we may have the situation in which the codeword itself is self-similar. Thus we introduce a new combinatorial object called synchronization circle, which enhances the non-self-similar property of a synchronization string and handles the corner case where the codeword is self-similar. Concatenate a synchronization circle with each codeword, we get a deterministic construction.

Now the task is essentially finding good codes for insdel codes with block length roughly $\log n$. As in [10], a good insdel codes can be constructed
using an efficient standard error correcting code and a synchronization string, which can be replaced with a synchronization circle. Now all we need is a synchronization circle of length roughly $\log n$ which can be constructed through brute force which takes polynomial time.

**Theorem 1.3.** An $\varepsilon$-synchronization string of length $n$ over alphabet of size $O(\varepsilon^{-3})$ can be constructed deterministically in $\text{poly}(n)$.

Notice that in the construction above, we only need an insdel code of block length $\log n$ and a synchronization circle of length $\log n$. We can recursively construct the synchronization circle of length $\log n$ with synchronization circle and insdel code of length $\log \log n$. Thus we can achieve a near linear time construction, but over an alphabet of size $\varepsilon^{-4}$.

**Theorem 1.4.** For $\varepsilon \geq C \left( \frac{(\log \log n)^2}{\log n} \right)$ for some constant $C > 1$ and $n \in \mathbb{N}$, an $\varepsilon$-synchronization string of length $n$ over alphabet of size $O(\varepsilon^{-4})$ can be constructed in near linear time.

## 2 Definitions and Preliminaries

### 2.1 Notations

- $S_i$ denotes the $i$th symbol of string $S$.

- $S[i, j]$ denotes the substring from $i$th position to $j$th position, both ends included; $S[i, j)$ denotes the substring from $i$th position to $j$th position, $S_i$ included, $S_j$ excluded.
• $|S|$ denotes the length of string $S$.

• $|\Sigma|$ denotes the size of alphabet $\Sigma$.

• $[n, k, d]$-error correcting code (ECC) denotes an error correcting code with block length $n$, message length $k$ and distance $d$.

• $|C|$ denotes the size of a code $C$, namely the number of codewords in $C$.

### 2.2 Definitions

**Definition 2.1** (Substring). A substring is the continuous symbols within a string $S$.

**Definition 2.2** (Subsequence). The subsequence of a string $S$ is any sequence of symbols obtained from $S$ by deleting some symbols. It doesn't have to be continuous.

**Definition 2.3** (Edit distance). For every $n \in \mathbb{N}$, the edit distance $ED(S, S')$ between two strings $S, S' \in \Sigma^n$ is the minimum number of insertions and deletions required to transform $S$ into $S'$.

**Definition 2.4** (Longest Common Subsequence). For any strings $S, S'$ over $\Sigma$, the longest common subsequence of $S$ and $S'$ is the longest pair of subsequence that are equal as strings. We denote by $LCS(S, S')$ the length of the longest common subsequence of $S$ and $S'$. 
Note that $ED(S, S') = |S| + |S'| - 2LCS(S, S')$ where $|S|$ denotes the length of $S$.

**Definition 2.5** ($\varepsilon$-synchronization circle). A string $S$ is an $\varepsilon$-synchronization circle if $\forall 1 \leq i \leq n, S_i, S_{i+1}, \ldots, S_n, S_1, S_2, \ldots, S_{i-1}$ is still an $\varepsilon$-synchronization string.

### 3 $\varepsilon$-synchronization Strings and Circles with Alphabet Size $O(\varepsilon^{-2})$

We now introduce a random algorithm constructing $\varepsilon$-synchronization string. First we show the existence of $\varepsilon$-synchronization string over alphabet of size $O(\varepsilon^{-2})$ with non-uniform sample space and Lovász Local Lemma. This would give a natural random algorithm using algorithmic Lovász Local Lemma in [16] and the extension in [9].

Then we introduce a simple construction of synchronization circles using synchronization strings. We’ll see later, that a synchronization string may violates the requirements of a synchronization circle only at the beginning and ending. We’ll concatenate two synchronization string over two different alphabet to construct synchronization circle.

#### 3.1 Existence and Random Construction of Synchronization String

Recall the general Locász Lemma:
Lemma 3.1. (General Lovász Local Lemma) Let $A_1, ..., A_n$ be a set of bad events. $G(V, E)$ is a dependency graph for this set of events if $V = \{1, \ldots, n\}$ and each event $A_i$ is mutually independent of all the events $\{A_j : (i, j) \notin E\}$.

If there exists $x_1, ..., x_n \in [0, 1)$ such that for all $i$ we have

$$\Pr(A_i) \leq x_i \prod_{(i, j) \in E} (1 - x_j)$$

Then the probability that none of these events happens is bounded by

$$\Pr[\bigwedge_{i=1}^{n} \bar{A}_i] \geq \prod_{i=1}^{n} (1 - x_i) > 0$$

Theorem 3.2. There exists an $\varepsilon$-synchronization string $S$ of length $n$ over alphabet $\Sigma$ of size $\Theta(\varepsilon^{-2})$ for any $\varepsilon \in (0, 1)$.

Proof: Suppose $|\Sigma| = c_1\varepsilon^{-2}$ where $c_1$ is a constant. Let $t = c_2\varepsilon^{-2}$ where $0 < c_2 < c_1$. The algorithm of constructing $S$ is as follows:

1. Randomly pick $t$ different symbols from $\Sigma$ without replacement and let them be the first $t$ symbols of $S$. If $t \geq n$, we just pick $n$ different symbols.

2. For $t + 1 \leq i \leq n$, we pick the $i$th symbol $S[i]$ uniformly randomly from $\Sigma \setminus \{S[i - 1], \ldots, S[i - t + 1]\}$. Namely, uniformly randomly pick the $i$th symbol conditioned that it’s different from previous $t - 1$ symbols.

To be a synchronization string, all the intervals $S[i, k)$ of $S$ should satisfies the condition that $ED(S[i, j), S[j, k)) > (1 - \varepsilon)(k - i)$, which is equivalent
to $LCS(S[i,j], S[j,k]) < \frac{\varepsilon}{2}(k - i)$. We define the badness of intervals as series of bad events. More specifically, denote an interval $S[i,k]$ as bad if $\exists i \leq j \leq k$ such that $LCS(S[i,j], S[j,k]) \geq \frac{\varepsilon}{2}(k - i)$.

To follow the Lovász Local Lemma, we need to bound the probability for an interval $S[i,k]$ being bad. According to the construction, the probability for intervals $S[i,k]$ where $k - i \leq t$ being bad is zero, because all symbols within such intervals are distinct and cannot be bad. Thus we only need to consider the intervals in which $k - i > t$. If an interval is bad, then there exists a repeating subsequence $a_1a_2\ldots a_ma_1a_2\ldots a_m$ where $m$ is at least $\frac{\varepsilon}{2}(k - i)$. Such sequence can be specified via $\varepsilon(k - i)$ positions in $S[i,k]$ and the probability that a given fixed sequence is valid for a random string is at most $(|\Sigma| - t)^{-\frac{1}{2}(k - i)}$. As the $i$th symbol should be different from previous $t - 1$ (or $i - 1$ if $i \leq t$) symbols, which means the size of the alphabet from which the $i$th symbol can be chosen from is at least $(|\Sigma| - t)$. Hence,

$$\Pr[\text{interval } S[i,k] \text{ is bad}] \leq \left(\frac{k - i}{\varepsilon(k - i)}\right)(|\Sigma| - t)^{-\frac{\varepsilon(k - i)}{2}}$$

Utilizing Stirling’s inequality, we have

$$\Pr[\text{interval } S[i,k] \text{ is bad}] \leq \left(\frac{k - i}{\varepsilon(k - i)}\right)(|\Sigma| - t)^{-\frac{\varepsilon(k - i)}{2}} \leq \left(\frac{e(k - i)}{\varepsilon(k - i)}\right)^{\varepsilon(k - i)}(|\Sigma| - t)^{-\frac{\varepsilon(k - i)}{2}} \leq \left(\frac{\varepsilon \sqrt{|\Sigma| - t}}{e}\right)^{-\varepsilon(k - i)} = C^{-\varepsilon(k - i)}$$
The resulting inequality shows that the probability of \( S[i, k] \) being bad is bounded by \( C^{-\varepsilon(k-i)} \) where \( C \) can be made arbitrarily large by taking suitable \( c_1 \) and \( c_2 \).

To show that there is a non-zero probability for \( S \) being an \( \varepsilon \)-synchronization string, we use the general Lovász Local Lemma in Lemma 3.1. Note that the lemma requires some mutually independence of bad events, we have the lemma below.

**Lemma 3.3.** The badness of interval \( I = S[i, j] \) is mutually independent of the badness of all intervals that do not intersect with \( I \).

**Proof of lemma 3.3:** To make the proof more clear, we use the following notations:

- \( I_1, \ldots, I_m \): the intervals before \( I \) that do not intersect with \( I \).
- \( I'_1, \ldots, I'_m \): the intervals after \( I \) that do not intersect with \( I \).
- \( b \): indicator of the badness of interval \( I \).
- \( b_k \): indicator of the badness of interval \( I_k \).
- \( b'_{k'} \): indicator of the badness of interval \( I'_{k'} \).

Now we prove that there exists \( p \in (0, 1) \) such that \( \forall x_1, x_2, \ldots, x_m \in \{0, 1\} \):

\[
\Pr[b = 1|b_k = x_k, k = 1, \ldots, m] = p
\]

According to the construction, the sampling of the symbols in \( S[i, k] \) only depends on previous \( t - 1 \) symbols (or \( i - 1 \) symbols is \( i < t \)). After fixing the
prefix $S[1, i-1]$, the size of the alphabet that each symbol of $I$ can be chosen from is fixed. Thus, conditioned on fixed prefix $S[1, i-1]$, the probability of $I$ being bad is a fixed real number $p'$. That is,

$$\forall \text{ valid } \tilde{S} \in \Sigma^{i-1}, \Pr[b = 1 | S[1, i-1] = \tilde{S}] = p'$$

Thus we have

$$\Pr[b = 1 | b_k = x_k, i = 1, \ldots, m] = \frac{\Pr[b = 1, b_k = x_k, i = 1, \ldots, m]}{\Pr[b_k = x_k, k = 1, \ldots, m]} = \sum_{\tilde{S}} \frac{\Pr[b = 1, S[1, i-1] = \tilde{S}]}{\sum_{\tilde{S}} \Pr[S[1, i-1] = \tilde{S}]}$$

$$= \sum_{\tilde{S}} \frac{(\Pr[b = 1, S[1, i-1] = \tilde{S}] \Pr[S[1, i-1] = \tilde{S}])}{\sum_{\tilde{S}'} \Pr[S[1, i-1] = \tilde{S}']}$$

$$= p' \sum_{\tilde{S}} \frac{\Pr[S[1, i-1] = \tilde{S}]}{\sum_{\tilde{S}'} \Pr[S[1, i-1] = \tilde{S}']}$$

In the equations, $\tilde{S}$ indicates all valid strings such that for prefix $S[1, i-1]$, $b_k = x_k, k = 1, \ldots, m$. Therefore, $b$ is independent of $\{b_k, k = 1, \ldots, m\}$.

Similarly we can show that the joint distribution of $\{b'_k, k' = 1, \ldots, m'\}$ is independent of $\{b, b_k, k = 1, \ldots, m\}$. Obviously $b$ is mutually independent of $\{b_k, b'_k, k = 1, \ldots, m, k' = 1, \ldots, m'\}$. Thus the badness of interval $I$ is
mutually independent from that of all intervals that do not intersect with \( I \).
\( \square \).

The rest of the proof is the same as that of Theorem 5.7 in [10].

Notice that an interval of length \( l \) intersects at most \( l + l' \) intervals of length \( l' \). We only need to find a sequence of real numbers \( x_{i,k} \in [0, 1) \) for intervals \( S[i,k] \) such that

\[
\Pr[S[i,k] \text{is bad}] \leq x_{i,k} \prod_{S[i,k] \cap S[i',k') \neq \emptyset} (1 - x_{i',k'})
\]

We propose \( x_{i,k} = D^{-\varepsilon(k-i)} \) for some constant \( D \geq 1 \). Hence we only need to find a constant \( D \) such that for all \( S[i,k] \),

\[
C^{-\varepsilon(k-i)} \leq D^{-\varepsilon(k-i)} \prod_{l=t}^{n} [1 - D^{-\varepsilon}]^{l+(k-i)}
\]

That is, for all \( l' \in \{1, ..., n\} \),

\[
C^{-l'} \leq D^{-l'} \prod_{l=t}^{n} [1 - D^{-\varepsilon}]^{\frac{l+l'}{\varepsilon}}
\]

which means that

\[
C \geq \frac{D}{\prod_{l=t}^{n} [1 - D^{-\varepsilon}]^{\frac{l+l'}+1}}
\]

Notice that the righthand side is maximized when \( n = \infty, l' = 1 \), it’s sufficient to show that

\[
C \geq \frac{D}{\prod_{l=t}^{\infty} [1 - D^{-\varepsilon}]^{\frac{l+1}}}
\]
Let \( L = \max_{D > 1} \frac{D}{\prod_{l=t}^{\infty} [1 - D^{-el}]^{\frac{l+1}{\epsilon}}} \). We only need to guarantee that \( C > L \), which can be done by modified \( c_1 \) and \( c_2 \).

We claim that \( L = \Theta(1) \). Since that \( t = c_2 \epsilon^{-2} = \omega \left( \frac{\log^\frac{1}{\epsilon} 2}{\epsilon} \right) \),

\[
\begin{align*}
\frac{D}{\prod_{l=t}^{\infty} [1 - D^{-el}]^{\frac{l+1}{\epsilon}}} & < \frac{D}{\prod_{l=t}^{\infty} [1 - l \frac{1}{\epsilon} D^{-el}]} \\
& < \frac{D}{1 - \sum_{l=t}^{\infty} l \frac{1}{\epsilon} D^{-el}} \\
& = \frac{D}{1 - \frac{1}{\epsilon} \sum_{l=t}^{\infty} (l + 1) D^{-el}} \\
& = \frac{D}{1 - \frac{D^{-\epsilon t}}{\epsilon (1 - D^{-\epsilon})^2}} \\
& = \frac{D}{1 - \frac{D^{-\epsilon t}}{\epsilon (1 - D^{-\epsilon})^2}}
\end{align*}
\]

Inequality (1) comes from the fact that \( (1 - x)^\alpha > 1 - \alpha x \), (2) comes from the fact that \( \prod_{i=1}^{\infty} (1 - x_i) \geq 1 - \sum_{i=1}^{\infty} x_i \) and (3) is a result from \( \sum_{l=t}^{\infty} (l + 1) x^l = \frac{x^l (1 + l - t x)}{(1 - x)^2} < \frac{2t x^t}{(1 - x)^2}, x < 1 \).

We can see that for \( D = 7 \), \( \max_{\epsilon} \{ \frac{2 \epsilon}{\epsilon (1 - D^{-\epsilon})^2} \} < 0.9 \). Therefore (5) is bounded by a constant, which means that \( L = \Theta(1) \) and the proof is complete.

### 3.2 Synchronization circle

We now construct an \( \epsilon \)-synchronization circle using Theorem 3.2. A synchronization circle is actually a generalization and strengthening of a synchronization string in the sense that no matter how we rotate the string,
the resulted string is still a synchronization string. A synchronization string would violates this requirement in intervals crossing the beginning of the original string. Thus we use two synchronization strings over two disjoint alphabets to overcome this.

**Theorem 3.4.** There exists an $\varepsilon$ synchronization circle $S$ of length $n$ over alphabet $\Sigma$ of size $\Theta(\varepsilon)^{-2}$ for any $\varepsilon \in (0, 1), n \in \mathbb{N}$.

**Proof:** Suppose we have two synchronization string over two completely different alphabets: $S_1$ with length $\lceil \frac{n}{2} \rceil$ over $\Sigma_1$ and $S_2$ with length $\lfloor \frac{n}{2} \rfloor$ over $\Sigma_2$. Let $\Sigma_1 \cap \Sigma_2 = \emptyset$ and $|\Sigma_1| = |\Sigma_2| = \Theta(\varepsilon^{-2})$. Let $S$ be the concatenation of $S_1$ and $S_2$. Then the original string $S = s_1s_2 \ldots s_n$ is over alphabet $\Sigma = \Sigma_1 \cup \Sigma_2$ whose size is $\Theta(\varepsilon^{-2})$. Now we prove that $S$ is an $\varepsilon$-synchronization circle.

For all $1 \leq m \leq n$, consider the rotated string $S' = s_m, s_{m+1}, \ldots, s_n, s_1, s_2, \ldots, s_{m-1}$.

Notice that for any two strings $s$ and $s'$, $\text{LCS}(s, s') \leq \frac{\varepsilon}{2}(|s| + |s'|)$ is equivalent to $\text{ED}(s, s') \geq (1-\varepsilon)(|s| + |s'|)$. For any $i < j < k$, we call an interval $S'[i, k)$ good if $\text{LCS}(S'[i, j), S'[j, k)) \leq \frac{\varepsilon}{2}(k-i)$. It suffices to show that for all $1 \leq i, k \leq n$, the interval $S'[i, k)$ is good.

If $m \geq \lceil \frac{n}{2} \rceil$, then intervals which are substrings of $S'[n-m+2, n-m+1+\lceil \frac{n}{2} \rceil] = S[1, \lfloor \frac{n}{2} \rfloor]$, $S'[1, n-m+1] = S[m, n]$ and $S'[n-m+1+\lceil \frac{n}{2} \rceil, n] = S[\lceil \frac{n}{2} \rceil, m-1]$ are good intervals as they are also substring of $S_2$ and $S_1$. We are now left with the intervals crossing $s_n$ or $s_{\lceil \frac{n}{2} \rceil}$.

**If $S'[i, k]$ crosses $s_n, s_1$ but doesn’t cross $s_{\lceil \frac{n}{2} \rceil}$:** If $j < n-m+1$, then there’s no common subsequence between $s'[i, j]$ and $S'[n-m+2, k]$, because
the symbols come from two disjoint alphabets. Thus

\[
LCS(S'[i, j], S'[j+1, k]) \leq LCS(S'[i, j], S'[j+1, n-m+1]) \leq \frac{\varepsilon}{2}(n-m+1-i) < \frac{\varepsilon}{2}(k-i)
\]

If \( j \geq n-m+1 \), then there’s no common subsequence between \( S'[j+1, k] \) and \( S'[i, n-m+1] \). Thus

\[
LCS(S'[i, j], S'[j+1, k]) \leq LCS(S'[n-m+2, j], S'[j+1, k]) \leq \frac{\varepsilon}{2}(k-(n-m+2)) < \frac{\varepsilon}{2}(k-i)
\]

Hence intervals of this kind are good.
Figure 1: Example where $S'[i, k]$ contains $s_n, s_1$ but doesn’t contain $s_{\lfloor \frac{n}{2} \rfloor}$

If $S'[i, k]$ contains $s_{\lfloor \frac{n}{2} \rfloor}, s_{\lceil \frac{n}{2} \rceil}$ but doesn’t contain $s_n$: If $j \leq n - m + \lfloor \frac{n}{2} \rfloor + 1$, then there’s no common subsequence between $S'[i, j]$ and $S'[n - m + \lceil \frac{n}{2} \rceil + 1, k]$, thus

$LCS(S'[i, j], S'[j + 1, k]) \leq LCS(S'[i, j], S'[j + 1, n - m + \lfloor \frac{n}{2} \rfloor + 1]) < \frac{\varepsilon}{2}(k - i)$

If $j \geq n - m + \lceil \frac{n}{2} \rceil + 1$, then there’s no common subsequence between
\[ S'[j+1, k] \text{ and } S'[i, n-m + \left\lfloor \frac{n}{2} \right\rfloor + 1]. \] Thus

\[ LCS(S'[i, j], S'[j+1, k]) \leq LCS(S'[n-m + \left\lfloor \frac{n}{2} \right\rfloor + 1, j], S'[j+1, k]) < \frac{\varepsilon}{2}(k-i) \]

Thus intervals of this kind are good.

**Figure 2:** Example where \( S'[i, k] \) contains \( s_{\left\lfloor \frac{n}{2} \right\rfloor}, s_{\left\lceil \frac{n}{2} \right\rceil} \)

**If \( S'[i, k] \) contains \( s_{\left\lfloor \frac{n}{2} \right\rfloor} \) and \( s_{n} \):** If \( n-m+2 \leq j \leq n-m + \left\lfloor \frac{n}{2} \right\rfloor + 1 \), then the common subsequence is either that of \( S'[i, n-m+1] \) and \( S'[n-m + \left\lfloor \frac{n}{2} \right\rfloor + 1, k] \) or that of \( S'[n-m+2, j] \) and \( S'[j+1, n-m + \left\lfloor \frac{n}{2} \right\rfloor + 1] \). This follows the fact
that, there’s a match between $S'[i, n-m+1]$ and $S'[n-m+\lceil \frac{n}{2} \rceil +1, k]$, then this common subsequence cannot contain matches between $S'[n-m+2, j]$ and $S'[j+1, n-m+\lceil \frac{n}{2} \rceil +1]$. Thus

\[
LCS(S'[i, j], S'[j +1, k]) \\
\leq \max\{LCS(S'[i, n-m+1], S'[n-m+\lceil \frac{n}{2} \rceil +1, k]), \\
LCS(S'[n-m+2, j], S'[j+1, n-m+\lceil \frac{n}{2} \rceil +1])\} \\
< \frac{\varepsilon}{2}(k-i)
\]

Figure 3: Example where $S'[i, k]$ contains $s_{\lfloor \frac{n}{2} \rfloor}$ and $s_n$
If \( j \leq n - m + 1 \), similarly then there’s no common subsequence between \( S'[i, j] \) and \( S'[n - m + 2, n - m + \lfloor \frac{n}{2} \rfloor + 1] \). Thus

\[
LCS(S'[i, j], S'[j + 1, k]) \\
\leq LCS(S'[i, j], S'[j + 1, n - m + 1]) + LCS(S'[n - m + \lceil \frac{n}{2} \rceil + 1, k]) \\
< \frac{\varepsilon}{2}(k - i)
\]

If \( j \geq S'[n - m + \lceil \frac{n}{2} \rceil + 1] \), the proof is similar with the case where \( j \leq n - m + 1 \). Hence \( S' \) is still an \( \varepsilon \)-synchronization string.

Similarly we know that when \( m \leq n - m + \lfloor \frac{n}{2} \rfloor + 1 \), \( S' \) is still an \( \varepsilon \)-synchronization string, which indicates that \( S \) is an \( \varepsilon \)-synchronization circle.

\[\blacksquare\]

## 4 Deterministic Constructions

We now construct \( \varepsilon \)-synchronization strings using synchronization circles. As mentioned in introduction, a synchronization string can be constructed by concatenation of codewords of an insdel code. We’ll show that the result of such concatenation is stronger: it’s a synchronization circle.

First recall the following result from [10].

**Lemma 4.1** (Theorem 4.2 of [10]). An \( \varepsilon \)-synchronization string along with an efficient ECC \( C \) over alphabet \( \Sigma_C \), that corrects up to \( n\delta + 2k \) half-errors can be used to construct an insertion/deletion code that can be decoded from up to \( n\delta \) deletions.
We now use the codewords from the insdel code in this lemma.

**Algorithm 1. Input:**

1. A ECC $\tilde{C} \subset \Sigma_\epsilon^m$, with distance $\delta m$ and $|\tilde{C}| \geq \ell = \lceil \frac{n}{m} \rceil$.
2. An $\epsilon$-synchronization circle $SC$ of length $m$ over alphabet $\Sigma_{sc}$ i.e. $SC = (sc_1, sc_2, \ldots, sc_m) \in \Sigma_{sc}^m$.

**Operations:**

- Construct a new code $C \subset \Sigma^m$ s.t.

  $C = \{c = ((\tilde{c}_1, sc_1), (\tilde{c}_2, sc_2), \ldots, (\tilde{c}_m, sc_m))|(\tilde{c}_1, \tilde{c}_2, \ldots, \tilde{c}_m) \in \tilde{C}\}$

  where $\Sigma = \Sigma_\epsilon \times \Sigma_{sc}$.

- Choose $\ell$ codewords $C_1, C_2, \ldots, C_\ell$ from $C$.

- Let $S$ be concatenation of these codewords: $S = C_1 \circ C_2 \circ \cdots \circ C_\ell$.

**Output:** $S$.

**Lemma 4.2.** The output $S$ in Algorithm 1 is an $\epsilon'$-synchronization circle, where $\epsilon' \leq 16(1 - \frac{1-\epsilon}{1+\epsilon}\delta)$.

**Proof:** As the distance of $\tilde{C}$ is $\delta m$, $\tilde{C}$ can correct up to $\delta m$ half errors. According to Lemma 4.1, together with $SC$, we can get an insdel code $C$ which can correct up to $\frac{1-\epsilon}{1+\epsilon}\delta m$ deletions.

Let $\alpha = 1 - \frac{1-\epsilon}{1+\epsilon}\delta$. $C$ has the following properties:
1. $LCS(\mathcal{C}) = \max_{c_0, c_1 \in \mathcal{C}} \{LCS(c_0, c_1)\} \leq \alpha m$

2. Each codeword in $\mathcal{C}$ is an $\varepsilon$-synchronization circle itself.

We only need to prove that $\forall 1 \leq i < j < k \leq n, LCS(S[i, j), S[j, k)) < \frac{\varepsilon}{2}(k - i)$. First we prove the lemma below.

**Lemma 4.3.** Suppose $S_1$ and $S_2$ are concatenation of different codewords from $\mathcal{C}$. $S_1 = c_1 \circ \cdots \circ c_{\ell_1}, S_2 = c'_1 \circ \cdots \circ c'_{\ell_2}$ where these codewords are all from $\mathcal{C}$ and are completely different. Then $LCS(S_1, S_2) \leq (\ell_2 + 2\ell_1)\alpha m$.

**Proof of 4.3:** Suppose the longest common subsequence between $S_1$ and $S_2$ is $\tilde{S}$. Consider $\tilde{S}$ as a match between $S_1$ and $S_2$. Let $j_i$ be the length of the shortest substring containing the subsequence matched by the $i$th codeword $c_i$ in $S_1$. See Figure 4 as an example in which $j_1 = 3, j_2 = 5, j_3 = 4$.

$$
\begin{array}{c|cccc}
S_1 & 1 & 2 & 3 & 5 & 6 \\
\hline
S_2 & 2 & 5 & 6 & 3 & 1 \\
\end{array}
\begin{array}{c|cccc}
\hline
j_1 & 1 & 2 & 3 & 5 & 6 \\
\hline
j_2 & 2 & 5 & 4 & 1 \\
\hline
j_3 & 3 & 4 & 5 & 1 & 2 \\
\end{array}
$$

Figure 4: Example of match between $S_1$ and $S_2$

Notice that $\sum_{i=1}^{\ell_1} j_i \leq \ell_2 m$. The number of codewords that intersect with the continuous substring matching with $c_i$ is at most $\lceil \frac{\varepsilon}{m} \rceil + 1$. Thus the length of the common subsequence $i$th codeword matches to is at most
\((\lceil \frac{j_1}{m} \rceil + 1)\alpha m.\)

\[
LCS(S_1, S_2) \leq \sum_{i=1}^{\ell_1} (\lceil \frac{j_i}{m} \rceil + 1)\alpha m
\leq \sum_{i=1}^{\ell_1} (\frac{j_i}{m} + 2)\alpha m
= \sum_{i=1}^{\ell_1} j_i \alpha + 2 \sum_{i=1}^{\ell_1} \alpha m
= (\ell_2 + 2\ell_1)\alpha m
\]

Thus \(LCS(S_1, S_2)\) is at most \((\ell_2 + 2\ell_1)\alpha m.\)

Suppose \(S_1 = S[i, j]\) and \(S_2 = S[j, k]\).

**If \(k - i > m, m \mid j\):** In this situation, \(S_j\) is exactly the end of a codeword. Let \(\ell_1\) and \(\ell_2\) denote the number of complete codewords in \(S_1\) and \(S_2\). If \(\ell_1 + \ell_2 = 0\), which means that there’s no complete codeword in both \(S_1\) and \(S_2\), then \(LCS(S_1, S_2) \leq \alpha m \leq \frac{\alpha}{2}(k - i)\). See example in Figure 5.

\[
\begin{array}{cccc}
\text{c}_1 & \text{c}_2 & \text{c}_3 \\
S & 1 & 2 & 3 & 5 & 6 & 1 & 5 & 6 & 2 & 4 & 3 & 1 & 4 & 5 & 6 \\
i & j & k
\end{array}
\]

Figure 5: Example in which \(k - i > m, m \mid j\) and \(\ell_1 + \ell_2 = 0\)

If \(\ell_1 + \ell_2 \geq 1\), except from these complete codewords, there may be incomplete codewords \(c_{\text{head}} = S[i, j - \ell_1 m - 1]\) in \(S_1\) and \(c_{\text{tail}} = S[j + \ell_2 m + 1, k]\) in \(S_2\). As shown in Figure 6.
\[
\begin{array}{cccc}
S & 1 & 2 & 3 & 5 & 6 \\
\hline
i & j & k \\
\hline
\text{c} & \text{head} & \text{c} & \text{tail} \\
\end{array}
\]

Figure 6: Example in which \( k - i > m, m \uparrow j, \ell_1 = 0, \ell_2 = 1 \)

In this case,

\[
\begin{align*}
LCS(S_1, S_2) \\
\leq & LCS(c_{\text{head}}, S_2) + LCS(S[j - \ell_1 m, j], S[j + 1, j + \ell_2 m + 1]) + LCS(S_1, c_{\text{tail}}) \\
\leq & ((\ell_2 + 1)am + (\ell_2 + 2\ell_1)am + (\ell_1 + 1)am \\
\leq & (2\ell_2 + 3\ell_1 + 2)am \\
\leq & 5(\ell_1 + \ell_2)am \leq 5\alpha(k - i)
\end{align*}
\]

\textbf{If} \( k - i > m, m \uparrow j \): In this situation, \( S_j \) splits a codeword which we denote as \( c_0 \). Suppose the part in \( S_1 \) is \( c_0^1 \) and that in \( S_2 \) is \( c_0^2 \). Notice that \( LCS(c_0^1, c_0^2) \leq \frac{\varepsilon_2}{32} m \), according to the property of synchronization circle.

Similarly with Case1, let \( \ell_1 \) and \( \ell_2 \) be the number of complete codewords in \( S_1 = S[i, j] \) and \( S_2 = S[j + 1, k] \). If \( \ell_1 + \ell_2 = 0 \), then \( LCS(S_1, S_2) \leq 4am \leq \frac{\varepsilon}{2}(k - i) \). As in Figure 7.

\[
\begin{array}{cccc}
S & 1 & 2 & 3 & 5 & 6 \\
\hline
i & j & k \\
\hline
\text{c} & \text{c} & \text{c} & \text{c} \\
\end{array}
\]

Figure 7: Example in which \( k - i > m, m \uparrow j, \ell_1 + \ell_2 = 0 \)
Now assume that \( \ell_1 + \ell_2 \geq 1 \). We denote the substring of \( S_1 \) from the first symbol to the beginning of \( c_0^1 \) with \( S'_1 \) and the substring of \( S_2 \) from the end of \( c_0^2 \) to the end as \( S'_2 \). If the longest common subsequence contains a match between \( c_0^1 \) and \( c_0^2 \), as in figure 8, then we have

\[
\begin{align*}
LCS(S_1, S_2) \\
\leq & \ LCS(c_0^1, c_0^2) + LCS(c_0^1, S'_2) + LCS(c_0^2, S'_1) \\
\leq & \alpha m + (\ell_2 + 1)\alpha m + (\ell_1 + 1)\alpha m \\
\leq & \ (\ell_1 + \ell_2 + 3)\alpha m \\
\leq & \ 4(\ell_1 + \ell_2)\alpha m \\
\leq & \ 4\alpha(k - i)
\end{align*}
\]

![Figure 8](image.png)

Figure 8: Example in which there’s match between \( c_0^1 \) and \( c_0^2 \)

If there’s no match between \( c_0^1 \) and \( c_0^2 \) in the longest common subsequence,
then the problem becomes similar to the first case, as in figure 9.

\[
LCS(S_1, S_2) \\
\leq LCS(S_1', c_0^2) + LCS(S_1' + S_2') + LCS(c_0^1, S_2) \\
\leq (\ell_1 + 1)am + (2\ell_2 + 3\ell_1 + 2)am + (\ell_2 + 1)am \\
\leq 8(\ell_1 + \ell_2)am \\
\leq 8\alpha(k - i)
\]

![Diagram](image)

Figure 9: Example in which there’s no match between \(c_0^1\) and \(c_0^2\)

**If \(k - i \leq m\):** In this situation, the longest common subsequence of \(S_1\) and \(S_2\) is no more than \(\frac{5}{2}(k - i) \leq am\) according to the property of synchronization circle.

Thus we have that, in any situation, the longest common subsequence of \(S[i, j]\) and \(S[j, k]\) is no more than \(8\alpha(k - i)\), which means that \(S\) is a \(16\alpha\)-synchronization string, that is, an \(\varepsilon'\)-synchronization string. As this proof applies for any shift of the start point of \(S\), \(S\) is an \(\varepsilon'\)-synchronization circle.
Now we fix the parameters and give the time complexity of the construction.

**Theorem 4.4.** An $\varepsilon$-synchronization string of length $n$ over alphabet of size $O(\varepsilon^{-3})$ can be constructed in $O(\varepsilon^{-1})^{O(\log n / \varepsilon)}\text{poly}(\log n / \varepsilon)$.

**Proof:** First, we need to construct a ECC $C$ with block length $m = O(\log n / \varepsilon)$ over alphabet $\Sigma_C$ of size $O(1/\varepsilon)$, with $n$ codewords and distance $d = \alpha m$ where $\alpha = \frac{1 + \frac{\varepsilon}{4\varepsilon}}{1 - \frac{\varepsilon}{16}}(1 - \frac{\varepsilon}{16}) = 1 - \Omega(\varepsilon)$. This would take time $O(n|\Sigma_C|^m m \log(1/\varepsilon))$ according to the following lemma.

**Lemma 4.5.** A ECC with block length $n$, number of codewords $N$, distance $d = (1 - \varepsilon)n$ and alphabet $\Sigma$ with size $O(1/\varepsilon)$ can be constructed in $O(N|\Sigma|^m n \log(1/\varepsilon))$.

**Proof of [Lemma 4.5]:** We conduct a brute force search to find all the codewords. The algorithm goes as follows:

1. Begin with $C = \emptyset$, $\Sigma_0 = \Sigma$.

2. for $i$ from 1 to $N$:
   
   (a) Add arbitrary element $c_i$ to $C$ from $\Sigma_{i-1}^n$.

   (b) Exclude elements with hamming distance less that $d$ from $C$. That is, $A_i = \{x \in \Sigma_{i-1} | x$ has distance less that $d$ from $C\}$ and $\Sigma_i = \Sigma_{i-1} \setminus A_i$. 

25
For a fixed $C \subset \Sigma^n$, the number of elements with distance less than $d$ from $C$ is at most $\binom{n}{d} |\Sigma|^d = \binom{n}{\xi n} |\Sigma|^{(1-\xi)n} \leq (\frac{\xi}{\xi})^{\varepsilon n} |\Sigma|^{(1-\varepsilon)n}$. Notice that here we have to ensure that $|C| (\frac{\xi}{\xi})^{\varepsilon n} |\Sigma|^{(1-\xi)n} = N (\frac{\xi}{\xi})^{\varepsilon n} |\Sigma|^{(1-\varepsilon)n} \leq |\Sigma|^n$, $|\Sigma| = O(1/\varepsilon)$.

In each exclusion procedure, we have to exhaustively search the whole space $\Sigma^n$ and compute the hamming distance to new added codeword $c_i$, which takes $O(n \log(1/\varepsilon))$. Thus the total running time of this algorithm is $O(N |\Sigma|^n n \log(1/\varepsilon))$.\□

Moreover, we can construct an $\frac{\varepsilon}{4\varepsilon}$-synchronization circle $SC$ of length $m$ over alphabet $\Sigma_{SC}$ of size $O(\varepsilon^{-2})$. Then according to the Algorithm 1 and Lemma 4.2, we have an $\varepsilon$-synchronization string $S$.

The time for constructing $SC$ is twice the time for constructing a synchronization string using Theorem 3.2. For each symbol we check the previous $t-1$ or $i-1$ symbols and uniformly randomly pick a different symbol. Thus the construction time for $SC$ is $O(\varepsilon^{-2})^m \text{poly}(m)$. The time constructing $S$ with $C$ and $SC$ takes $O(n)$. Thus the total running time is

$$O(n|\Sigma_C|^m n \log(1/\varepsilon)) + O(\varepsilon^{-2})^m \text{poly}(m) + O(n)$$

$$= O(\varepsilon^{-1})O\left(\frac{\log n}{\varepsilon}\right) \text{poly}\left(\frac{\log n}{\varepsilon}\right)$$

Regarding $\varepsilon$ as an constant, then the running time is $\text{poly}(n)$. $S$ is a synchronization string of length $n$ over alphabet $\Sigma = \Sigma_C \times \Sigma_{SC}$ of size $O(\varepsilon^{-3})$.
\□

To make the construction more efficient, we can recursively construct
synchronization strings to reduce the running time to $O(n \text{poly} \log n)$, which is near linear. Notice that according to Lemma 4.2, $S$ is a synchronization circle, we have the following theorem.

**Theorem 4.6.** An $\varepsilon$ synchronization circle $S$ of length $n$ over alphabet $\Sigma$ of size $O(\varepsilon^{-4})$ can be constructed in $O(n(\log \log n)^2)$. Here $n \in \mathbb{N}$ and $\varepsilon \geq \frac{C(\log \log \varepsilon)^2}{\log \varepsilon}$ where $C$ is a constant larger than 1.

**Proof:** The idea is simple: we first construct a synchronization circle of length $O(\log n)$ using Theorem 4.4. Then together with an error correcting code with block length $O(\log n)$, we have a synchronization circle of length $n$.

Formally, we construct an efficient error correcting code $C$ by concatenating an outer Reed-solomon code $RS$ and an inner code $C_0$. $RS$ is a $[m_1 = O(\frac{\log n}{\varepsilon}), \Omega(\varepsilon m_1), (1 - O(\varepsilon))m_1]$ code, while $C_0$ had block length $m_0 = O(\frac{\log m_1}{\varepsilon})$, alphabet size $O(1/\varepsilon)$, number of codewords $m_1$ and distance $(1 - O(\varepsilon)m_0)$. The inner code can be constructed using Lemma 4.5 within $m_1O(\frac{1}{\varepsilon}m_0 \log \log \frac{1}{\varepsilon})$.

The encoding time for $RS$ is as following lemma:

**Lemma 4.7.** A $(n, k, d)$ Reed-Solomon code over alphabet of size $O(n)$ can be encoded within $O(n \log^3 n)$.

**Proof of []:** Lemma 4.7] The encoding procedure of Reed-Solomon code can be regarded as multi-point evaluation of a polynomial function. Regard the message as the $k$ coefficients of a $k - 1$ degree polynomial $P$. From the result of [1, 15, 2], evaluating $P$ on $n$ points needs $O(n \log^2 n)$ arithmetic operations.
(+ and × on corresponding field). As the field has size $O(n)$, the addition operation takes $O(\log n)$, multiplication takes $O(\log^2 n)$, the total running time is thus $O(n \log^2 n)$. Thus encoding of RS needs $O(m_1 \log^3 m_1)$. The time to compute one codeword of $C$ is $O(m_1 \log^2 m_1 + m_1 m_0 \log(1/\varepsilon))$. Notice that only $\frac{n}{m_1 m_0}$ codewords from $C$ are needed to construct $S$, the time for constructing necessary codewords from $C$ is $\frac{n}{m_1 m_0} O(m_1 \log^2 m_1 + m_1 m_0 \log(1/\varepsilon))$.

According to Theorem 4.4, the time needed to construct a synchronization circle of length $m_0 m_1$ over alphabet of size $O(\varepsilon^{-3})$ is $O(\varepsilon^{-2}) O(\frac{\log(m_0 m_1)}{\varepsilon}) \text{poly}(\frac{\log(m_0 m_1)}{\varepsilon})$. And the time for copying it is $O(n \log(1/\varepsilon))$. Thus the total running time is

$$O(\varepsilon^{-2}) O(\frac{\log(m_0 m_1)}{\varepsilon}) \text{poly}(\frac{\log(m_0 m_1)}{\varepsilon}) \text{(time for constructing SC)}$$

$$+ O(n \log(1/\varepsilon)) \text{(time for copying SC)} + O(m_1 \log^2 m_1 + m_1 m_0 \log(1/\varepsilon)) \text{(time for computing inner codewords from C)}$$

$$+ \frac{n}{m_1 m_0} O(m_1 \log^2 m_1 + m_1 m_0 \log(1/\varepsilon)) \text{(time for computing codewords from C)}$$

$$= (O(\frac{1}{\varepsilon})) O(\frac{\log \log n + \log(1/\varepsilon)}{\varepsilon}) + O(n \log(1/\varepsilon)) + O(\varepsilon n \log^2 \frac{\log n}{\varepsilon})$$

If we pick $\varepsilon \geq \frac{C(\log \log n)^2}{\log n}$ for some constant $C > 1$, then the running time is $O(n (\log \log n)^2)$, which is near linear if regarding $\varepsilon$ as a constant.

### References


Biography

Ke Wu was born in 1994 in China.

Ke Wu finished undergraduate work at Fudan University in China majoring in Mathematics. During undergraduate study, she did some researches on SGD and its improvement algorithms.

In 2016, Ke Wu began her study in Computer Science of Johns Hopkins University. She was a research assistant for Prof. Xin Li.