Correcting a substring edit error of bounded length

Yuanyuan Tang*, Sarvin Motamen*, Hao Lou*, Kallie Whritenour§,
Shuche Wang†, Ryan Gabrys‡, and Farzad Farnoud*§

*Electrical & Computer Engineering, §Computer Science, University of Virginia, U.S.A.,
{yt5tz,awz4up,hl2nu,kw5km,farzad}@virginia.edu
†Institute of Operations Research and Analytics, National University of Singapore, shuche.wang@u.nus.edu
‡Calit2, University of California-San Diego, U.S.A., rgabrys@ucsd.edu

***Abstract*—Localized errors, which occur in windows with bounded lengths, are common in a range of applications. Such errors can be modeled as k-*substring edits*, which replace one substring with another string, both with lengths upper bounded by k. This generalizes errors such as localized deletions or burst substitutions studied in the literature. In this paper, we show through statistical analysis of real data that substring edits better describe differences between related documents compared to independent edits, and thus commonly arise in problems related to data synchronization. We also show that for the dataset under study, assuming codes exist that can achieve the Gilbert-Varshamov bound, substring-edit-correcting codes can synchronize two documents with much lower overhead compared to general indel/substitution-correcting codes. Furthermore, given a constant k, we construct binary codes of length n for correcting a k-substring edit with redundancy of roughly $2\log n$, compared to $8\log n$, the lowest redundancy achievable by an existing code for this problem. The time complexities of both encoding and decoding are polynomial with respect to n.**

I. Introduction

Localized errors are errors that cluster in windows with lengths much shorter than the whole sequence and are observed in various applications such as wireless communication, disk data storage, DNA storage, and document synchronization. The problem of burst or localized errors has been studied by several works [1]–[10]. More specifically, codes for correcting a burst of at most k substitutions were proposed in [1], [3]. Codes capable of correcting a burst of exactly k deletions were studied by [4]–[6], including [4], [5] over binary sequences and [6] over q-ary sequences. Furthermore, [5], [7], [10], [11] focused on correcting a burst of at most k deletions (or k insertions) while the works in [8], [9] studied localized deletions occurring in a window with bounded length k. The current paper focuses on correcting a k-*substring edit* error, which replaces one substring with another string at the same position, both with lengths bounded by k. Note that burst deletions, burst insertions, burst substitutions, and deletions/insertions occurring in a bounded window can all be considered as substring edits with a bounded length.

Unlike prior works that assume the prevalence of burst errors, we first present an experimental analysis to statistically evaluate this hypothesis. Based on two datasets, two versions of the source code of the bash shell and the DNA sequencing data exposed to errors, statistical tests provide strong evidence against the null hypothesis, namely that errors/edits are distributed uniformly in the sequences. Moreover, we study the suitability of substring-edit-correcting codes for error-correction and data synchronization. For each data sequence and its edited version, the experiment shows that substring-edit-correcting codes achieve lower redundancy than general indel/substitution-correcting codes.

Our problem is related to the problem of deletion-correcting codes and burst-deletion-correcting codes. However, as we will show in Lemma 1, a code that can correct a burst of at most ℓ deletions (or one that can correct a burst of at most ℓ insertions) cannot necessarily correct a k-substring edit, even if ℓ is much larger than k. On the other hand, a k-substring edit can be corrected by a code that can correct 2 bursts of at most k deletions and a code that can correct at most $2k$ deletions. These observations lead to the conclusion that existing codes for correcting multiple deletions [12], [13] or multiple bursts of deletions [14] can correct a k-substring edit with redundancy at least $8k\log n$ [13]. The goal of the current work is to construct codes that can correct a k-substring edit with redundancy roughly $2\log n$.

While codes for correcting a burst of at most k deletions cannot correct a k-substring edits, the idea of first identifying an approximate location for the error presented in Lenz et al. [7] and Bitar et al. [8] using the position of specific patterns is useful for our problem. We divide k-substring edits into *strict* k-substring edits (that will change the length in the outputs) and bursts of at most k substitutions, referred to as k-burst substitutions. For strict edits, we first extend the codes in [7], [8] to locate the error to be in an interval of length $O((\log n)^2)$ and then correct it. For k-burst substitutions, which cannot be located using the patterns mentioned above, we adapt the Fire code [1]. Then we combine the two error-correcting codes in a manner that enables polynomial-time encoding and decoding.

The paper is organized as follows. Section II presents the notation and preliminaries. Section III discusses the prevalence of burst errors in real data and the utility of substring-edit-correcting codes. Finally, Section IV presents the code constructions and an analysis of the time complexity and the

This work was supported in part by NSF grants under grant nos. CCF-1816409, CAREER-2144974, and CCF-2212437.

redundancy. Due to space limitations, the proofs are omitted.

II. Notation and Preliminaries

A. Notation

Without loss of generality, let $\Sigma_q = \{0, 1, \ldots, q-1\}$ be a finite alphabet of size q. The set of length-n strings and finite strings over Σ_q are denoted as Σ_q^n and Σ_q^*, respectively. The empty string, denoted Λ, is also considered a member of Σ_q^*. In this paper, we focus on the binary alphabet Σ_2. For $a, b \in \mathbb{Z}$, let $[a, b] = \{a, a+1, \ldots, b\}$ and $[b] = [1, b]$. Unless otherwise stated, logarithms are to the base of 2. For $\boldsymbol{x}, \boldsymbol{y} \in \Sigma_2^n$, let $\boldsymbol{x}_{[a,b]} = x_a x_{a+1} \cdots x_b$ and let $\boldsymbol{xy}$ and $(\boldsymbol{x}, \boldsymbol{y})$ denote the concatenation of $\boldsymbol{x}, \boldsymbol{y}$. For $\boldsymbol{x}, \boldsymbol{v} \in \Sigma_q^*$, $\boldsymbol{v}$ is a substring of $\boldsymbol{x}$ if $\boldsymbol{x} = \boldsymbol{uvw}$ for some $\boldsymbol{u}, \boldsymbol{w} \in \Sigma_q^*$. Furthermore, $|\boldsymbol{x}|$ is the length of a sequence $\boldsymbol{x}$ and $\|S\|$ is the number of elements in a set S. Given an integer r and a symbol $a \in \Sigma_2$, let a^r denote a *run* of r consecutive symbols a.

B. The k-substring edit channel

Given a string $\boldsymbol{x}$, a *k-burst deletion* (resp. *insertion*) in $\boldsymbol{x}$ removes (resp. inserts) at most k consecutive symbols, while a *k-substring edit* replaces a substring $\boldsymbol{v}$ of $\boldsymbol{x}$ by another string $\boldsymbol{v}'$, where $|\boldsymbol{v}|, |\boldsymbol{v}'| \leq k$ and at least one of $\boldsymbol{v}, \boldsymbol{v}'$ is non-empty. The k-substring edit is a *k-burst substitution* if $|\boldsymbol{v}| = |\boldsymbol{v}'|$ and a *strict* k-substring edit otherwise. In particular, $\boldsymbol{v}' = \Lambda$ results in a burst deletion addressed by previous works. For example, given $\boldsymbol{x} = 1001\underline{11011}1101 \in \Sigma_2^n$, a 4-substring edit may generate $\boldsymbol{z} = 1001\underline{010}1101$ by replacing $\boldsymbol{x}_{[5,8]} = 1101$ with $\boldsymbol{z}_{[5,7]} = 010$.

The next lemma discusses the relationship between deletion-correcting codes and codes that can correct a k-substring edit.

Lemma 1. *The codes in the statements below are over Σ_q^n, where $n, q \geq 2$. Let k be a positive integer.*

1) *A code that can correct 2 k-burst deletions can correct a k-substring edit.*
2) *For any $\ell < n$, there exists a code that can correct a burst of at most ℓ deletions but not a k-substring edit.*
3) *For any $\ell < n$, if $\ell < 2k$, then there exists a code that can correct up to ℓ deletions but not a k-substring edit.*

Given a code $\mathcal{C} \subseteq \Sigma_q^n$, the redundancy of the code $\mathcal{C}$ is defined as $n \log q - \log \|\mathcal{C}\|$. For binary, which is the focus of our code construction, part 1 of the above lemma implies the code given in [13] for correcting $\leq 2k$ deletions can correct a k-substring edit over the binary alphabet with the redundancy of roughly $8k \log n$ bits. To the best of our knowledge, that is the lowest redundancy that can be achieved by an existing code for this problem. The code we present in Theorem 19 has redundancy of roughly $2 \log n$.

Given a string $\boldsymbol{x} \in \Sigma_q^n$, let $D_{b,k}(\boldsymbol{x}) \subseteq \Sigma_q^*$ denote the set of strings generated from $\boldsymbol{x}$ by at most b k-substring edit errors and let $B_{b,k}(\boldsymbol{x}) \subseteq \Sigma_q^n$ denote the *confusable set* of $\boldsymbol{x}$, i.e., the set of sequences $\boldsymbol{y}$ other than $\boldsymbol{x}$ for which $D_{b,k}(\boldsymbol{x}) \cap D_{b,k}(\boldsymbol{y}) \neq \varnothing$. When $b = 1$ and k is clear from the context, we use $D(\boldsymbol{x}), B(\boldsymbol{x})$ instead of $D_{b,k}(\boldsymbol{x}), B_{b,k}(\boldsymbol{x})$.

We now find the *Gilbert-Varshamov (GV) bound* on the size of the code. Define $r_n(b, k) = \max_{\boldsymbol{x} \in \Sigma_q^n} \|B_{b,k}(\boldsymbol{x})\| + 1$.

Lemma 2. *Assuming an alphabet of size q, we have*

$$r_n(b, k) \leq (n + bk)^{2b} (k+1)^{4b} q^{2kb}$$

and there exists a code $\mathcal{C} \subseteq \Sigma_q^n$ of length n capable of correcting at most b k-substring edits with the size at least

$$\|\mathcal{C}\| \geq \frac{q^n}{(n + bk)^{2b} (k+1)^{4b} q^{2kb}}.$$

Assuming b, k are constants, the redundancy is bounded above by $2b \log n + o(\log n)$, which is the same as the redundancy of the codes proposed in this paper for $b = 1$ and the binary alphabet.

C. Relevant Prior Results

We first adapt a result from *syndrome compression*, a technique used to construct codes with low redundancy [15], to our problem.

Theorem 3 (c.f. [15, Theorem 5])**.** *Given $\boldsymbol{x} \in \Sigma_2^n$, let $f : \Sigma_2^n \to \Sigma_{2^{\mathcal{R}(n)}}$ be a (labeling) function over the confusable set $B(\boldsymbol{x})$ such that $f(\boldsymbol{x}) \neq f(\boldsymbol{y})$ for every $\boldsymbol{y} \in B(\boldsymbol{x})$, where $\mathcal{R}(n) = o(\log \log n \cdot \log n)$. Then there exists an integer $a \leq 2^{\log \|B(\boldsymbol{x})\| + o(\log n)}$ such that for all $\boldsymbol{y} \in B(\boldsymbol{x})$, we have $f(\boldsymbol{x}) \not\equiv f(\boldsymbol{y}) \bmod a$.*

We will use the following definitions and results from [7], [8]. These works correct a burst of deletions with low redundancy by first locating the approximate position of the error. Given sequences $\boldsymbol{x} \in \Sigma_2^n$ and a *pattern* (string) $\mathcal{P}$, define $\mathbf{1}_{\mathcal{P}}(\boldsymbol{x}) \in \Sigma_2^n$ as the indicator vector whose ith element is 1 if $\boldsymbol{x}_{[i,i+|\mathcal{P}|-1]} = \mathcal{P}$ and is 0 if $\boldsymbol{x}_{[i,i+|\mathcal{P}|-1]} \neq \mathcal{P}$ or $i + |\mathcal{P}| - 1 > n$. Further, let $n_{\mathcal{P}}(\boldsymbol{x})$ denote the number of 1's in $\mathbf{1}_{\mathcal{P}}(\boldsymbol{x})$ and $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})$ represent a length-$(n_{\mathcal{P}}(\boldsymbol{x}) + 1)$ vector whose ith element is the distance between positions of the $(i-1)$-th and the ith 1 in the string $(1, \mathbf{1}_{\mathcal{P}}(\boldsymbol{x}), 1)$. A sequence $\boldsymbol{x}$ is $(\mathcal{P}, \delta)$-dense if each interval of length δ in $\boldsymbol{x}$ contains at least one substring $\mathcal{P}$. The set of $(\mathcal{P}, \delta)$-dense binary strings of length n is denoted $\mathcal{D}_{\mathcal{P},\delta}(n)$. For $\mathcal{P} = 0^k 1^k$, $n \geq 5$, and $\delta = k 2^{2k+1} \lceil \log n \rceil$, we have [7]

$$|\mathcal{D}_{\mathcal{P},\delta}(n) \cap \Sigma_2^n| \geq 2^{n-1}. \tag{1}$$

Given a binary string $\boldsymbol{x} \in \Sigma_2^n$, define the *Varshamov-Tenengolts (VT) check sum* as $VT(\boldsymbol{x}) = \sum_{i=1}^n i x_i$. We next recall the code in [8] used to locate the burst of deletions or localized deletions in an interval.

Lemma 4 (cf. [8])**.** *Given integers $c_1 \in [0, 4]$, $c_2 \in [0, 6n-1]$, and $\delta = k 2^{2k+1} \lceil \log n \rceil$, there exists a code*

$$\begin{aligned} \mathcal{C}_d = \{ \boldsymbol{x} \in \Sigma_2^n \cap \mathcal{D}_{\mathcal{P},\delta}(n), n_{\mathcal{P}}(\boldsymbol{x}) = c_1 \mod 5, \\ VT(\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})) = c_2 \mod 6n. \} \end{aligned} \tag{2}$$

that can locate a burst of deletions or localized deletions in an interval with length $O((\log n)^2)$.

III. Substring edits in nanopore sequencing and document editing

In this section, we investigate the hypothesis that in real-world settings, errors/edits commonly occur in a bursty manner, rather than being distributed uniformly. We also study whether substring-edit-correcting codes can achieve lower redundancy than general edit-correcting codes. We performed experimental studies on two real-world datasets, corresponding to two applications of the codes:

- *Error-correction:* We investigate the set of errors encountered in nanopore sequencing [16] in DNA data storage. The data consists of 1000 input-output pairs, where the input represents the (true) base sequence of a DNA molecule, and the output represents the sequence detected by nanopore, as simulated by using the nanopore deep simulator [17] and the Nanopore's Guppy basecaller.
- *Data synchronization:* Suppose Alice, who knows only $\boldsymbol{x}$, needs to communicate $\boldsymbol{x}$ to Bob, who knows only $\boldsymbol{z}$, where $\boldsymbol{x}$ is the edited version of $\boldsymbol{z}$. This task is referred to as *data synchronization*, and can be accomplished using error-correcting codes [18]. The dataset consists of versions 5.0 and 5.1 of the source code for the Bash utility[1], containing 1301 and 1383 files, respectively, where each common file in version 5.1 is viewed as an edited version of the one in version 5.0.

Recall that our goal is to determine whether edits/errors are i) bursty or ii) uniformly/independently distributed. To rigorously answer this question, one way is to first define a uniform/independent random process for errors and edits and then use hypothesis testing to determine if such a model explains the observed data, which will be discussed in Subsection III-B. First, however, we perform an intuitive but less rigorous test over the alignment of pairs of sequences in Subsection III-A. Finally, in Subsection III-C, we consider choosing the model that leads to the lowest cost in error-correction and synchronization tasks for our data.

A. Independence test on alignment

Let $\boldsymbol{z}$ be the erroneous/edited version of $\boldsymbol{x}$. An alignment between $\boldsymbol{x}$ and $\boldsymbol{z}$ identifies the positions where the sequences match and how they differ (see Figure 1). From the alignment, let us produce the binary sequence, denoted $\boldsymbol{a}$, in which 1 represents a match and 0 represents an edit (substitution, insertion, deletion). If edits/errors are distributed in a uniform manner over $\boldsymbol{x}$, e.g., resulting from an "independent" process, then it is reasonable to expect the alignment $\boldsymbol{a}$ to resemble an iid sequence. Therefore, we apply the Wald-Wolfowitz runs-test [19].

```
G--TCATCCCG
|//|-|||||.
GGAT-ATCCCC
```

Figure 1: An alignment of two DNA sequences, where the top sequence can be obtained from the bottom one via deletions ($/$), insertions ($-$), and substitutions ($\cdot$).

[1]https://ftp.gnu.org/gnu/bash/

Here, the null hypothesis is that $\boldsymbol{a}$ is iid generated and the number R of runs in $\boldsymbol{a}$ is the test statistic. Conditioned on the number of 0's t_0 and the number of 1's t_1, the p-value is $\Pr(R \leq r | t_0, t_1)$, as we do not expect to see very few runs in an iid sequence. As in "RUNS-TEST" column of Table I, this test strongly suggests that edits are not iid for both datasets.

	RUNS-TEST	INS-TEST	SUBDEL-TEST
Bash	97.2%	100%	97.6%
DNA	95.7%	97.1%	76.1%

Table I: Fraction of sequences rejecting the null hypothesis at p-value threshold of 5%.

B. A probabilistic edit process

A more reasonable approach is to perform hypothesis testing on a probabilistic edit process. Again, let $\boldsymbol{x}$ be our data sequence of length n, and $\boldsymbol{z}$ its edited version. Based on an intuitive interpretation of "uniform edits", we define the following simple edit process: i) A random number K_i of insertions are uniformly distributed over the $n+1$ bins between x_i and x_{i+1}, $i = 0, \ldots, n$, where x_0 and x_{n+1} are defined as the empty symbol. ii) A random number K_d of substitutions and deletions are uniformly applied on $x_1, \ldots, x_n$.

The null hypothesis is that $\boldsymbol{z}$ is generated by this edit process. Since insertions occur independently of substitutions and deletions, we apply two separate tests. For insertions, we consider W, the number of non-empty bins as the test statistic. If most insertions cluster in a small number of bins, i.e., W being small, then we reject the null hypothesis. The p-value $\Pr(W \leq w | K_i)$ is summarized in "INS-TEST" column of Table I. For substitutions and deletions, we consider again R, the number of runs in the edit pattern (excluding insertions) as the test statistic, and reject the null hypothesis if R is small. The results are given in "SUBDEL-TEST" column of Table I. High rejection rates for both tests suggest that edits are not uniform.

C. Operational evaluation of error/edit models

Given two sequences $\boldsymbol{x} \in \Sigma_q^n$ and $\boldsymbol{z} \in \Sigma_q^*$, their differences can be described via b k-substring edits for a range of possible pairs (b, k). Operationally, the best description, i.e., (b, k) pair, is the one that leads to the lowest cost for the task at hand. For error-correction, where $\boldsymbol{z}$ is an erroneous copy of $\boldsymbol{x}$, the cost is the redundancy of the code that allows correcting the errors in $\boldsymbol{z}$. If $\boldsymbol{z}$ can be obtained from $\boldsymbol{x}$ via b k-substring edits, based on the GV bound, there exists such a code of length n with redundancy $\log r_n(b, k)$. For synchronization, the cost is the information exchange, i.e., the number of bits needed to be transmitted. It can be shown [18] that exchanging $\log r_n(b, k)$ bits is sufficient, achievable using a systematic code with n information symbols and $\log_q r_n(b, k)$ check symbols.

For each pair of sequences in the genome data set, among all valid (b, k) pairs, we find the one that minimizes $r_n(b, k)$, where $n = 200, q = 4$. The histogram of the best (b, k) pairs is given in Figure 2, which indicates that viewing the errors as 13 2-substring edits minimizes the redundancy for the largest

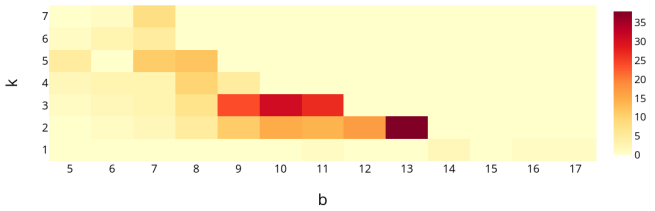

Figure 2: Histogram of optimal (b, k) values for the Nanopore sequencing dataset.

number of input-output pairs.[2]. This suggests that edits with $k > 1$ better describe our data and codes correcting b k-substring edits for $k > 1$ are of use in DNA data storage. The similar conclusion can also be drawn from the analysis of the bash dataset.

IV. Error-correcting codes for a k-substring edit

Given a constant k, this section focuses on constructing codes of length N for correcting a k-substring edit error with redundancy roughly $2\log N$ and polynomial time complexities in both the encoding and decoding processes. Unless otherwise stated, let n represent the length of $(\mathcal{P}, \delta)$-dense strings.

Based on Lemma 4, given an input $\boldsymbol{x} \in \mathcal{D}_{\mathcal{P},\delta}(n)$, locating the burst of deletions relies on the number of patterns $n_{\mathcal{P}}(\boldsymbol{x})$ and the relative distances of every two adjacent patterns $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})$. Compared with locating a burst of deletions, locating a k-substring edit is more complicated. Suppose $\boldsymbol{x} \in \mathcal{D}_{\mathcal{P},\delta}(n)$ and $\boldsymbol{y} \in D(\boldsymbol{x})$ is an output generated from $\boldsymbol{x}$ by a k-substring edit. We need to overcome the following challenges. First, when the k-substring edit is a substitution, i.e., $|\boldsymbol{x}| = |\boldsymbol{y}|$, it is possible for both $\boldsymbol{a}_{\mathcal{P}}$ and $n_{\mathcal{P}}$ to remain the same, preventing us from locating the error. Second, even if the substring edit is strict, i.e., $|\boldsymbol{x}| \neq |\boldsymbol{y}|$, there is no guarantee that the changes affecting $\boldsymbol{a}_{\mathcal{P}}$ and $n_{\mathcal{P}}$ will enable us to identify the approximate location of the error.

The following corollary summarizes our approach to construct error-correcting codes reaching the GV bound.

Corollary 5. *The code $\mathcal{C} \subseteq \Sigma_2^N$ is capable of correcting a k-substring edit if it can correct either a k-burst substitution or a strict k-substring edit error.*

A. Error-correcting code for a strict k-substring edit

Similar to works in [7], [8], given a constant k, this subsection focuses on correcting a strict k-substring edit by first localizing the error and then correcting it in the interval. More specifically, it consists of two steps. First, we extend the error-locating code in Lemma 4 from [8] to locate the strict k-substring edit in an interval of length $L = O((\log n)^2)$ with redundancy roughly $\log n$. Second, a modified *syndrome compression code* [10], [15] is designed to correct the error in the specific interval with redundancy roughly $O(\log\log n)$.

[2]We point out that only 272 sequence pairs are included as the rest are so erroneous that they have their minimum redundancy $\log r_n(b, k)$ larger than the original length (400 bits for $n = 200, q = 4$).

1) Locating the error in an interval: This subsection focuses on extending the codes in Lemma 4 to locate a strict k-substring edit since it will affect at least one of $n_{\mathcal{P}}(\boldsymbol{x})$ and $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})$.

Lemma 6. *Given $\mathcal{P} = 0^k1^k$ and $\delta = k2^{2k+1}\lceil\log n\rceil$, let $\boldsymbol{x} \in \mathcal{D}_{\mathcal{P},\delta}(n) \cap \Sigma_2^n$ be a $(\mathcal{P}, \delta)$-dense string and $\boldsymbol{y} \in D(\boldsymbol{x})$. Then a k-substring edit does not create nor destroy more than two* adjacent *patterns $\mathcal{P}$ in $\boldsymbol{x}$, i.e., $n_{\mathcal{P}}(\boldsymbol{y}) \in [n_{\mathcal{P}}(\boldsymbol{x})-2, n_{\mathcal{P}}(\boldsymbol{x})+2]$.*

Then we have the following corollary.

Corollary 7. *Let $\boldsymbol{x} \in \mathcal{D}_{\mathcal{P},\delta}(n)$ and $\boldsymbol{y} \in D(\boldsymbol{x})$. Then $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{y})$ can be generated from $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})$ as a result of a 3-substring-edit.*

According to the changes of $n_{\mathcal{P}}(\boldsymbol{x})$ and $\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})$, we extend the code in Bitar et al. work [8] as the following construction that helps to locate a strict k-substring edit in $\boldsymbol{x} \in \mathcal{D}_{\mathcal{P},\delta}(n)$.

Construction 8. *Given $0 \le c_1 \le 4$ and $0 \le c_2 < 7n$, let*

$$\mathcal{C}_{loc}(c_1, c_2) = \{\boldsymbol{x} \in \Sigma_2^n \cap \mathcal{D}_{\mathcal{P},\delta}(n), n_{\mathcal{P}}(\boldsymbol{x}) = c_1 \bmod 5, \\ VT(\boldsymbol{a}_{\mathcal{P}}(\boldsymbol{x})) = c_2 \bmod 7n\}.$$

Then we have the following lemma.

Lemma 9. *Let k be a constant. Given $\boldsymbol{x} \in \mathcal{C}_{loc}(c_1, c_2)$, a strict k-substring edit occurring in $\boldsymbol{x}$ can be located in a substring of $\boldsymbol{x}$ with length $L = O(\delta^2) = O((\log n)^2)$.*

2) Correcting the error in an interval: Suppose a strict k-substring edit is located in an interval of lenth $L = O(\delta^2)$. Next, we present error-correcting codes that can correct the strict k-substring edit in the specific interval with length L.

Similar to the work [10], we generate two sets of blocks of length $2L$ by partitioning $\boldsymbol{x} \in \mathcal{D}_{\mathcal{P},\delta}(n)$. More specifically, given $\hat{N} = n/2L$, let $S_e = (\boldsymbol{s}_{e1}, \boldsymbol{s}_{e2}, \cdots, \boldsymbol{s}_{e\hat{N}})$ and $S_o = (\boldsymbol{s}_{o1}, \boldsymbol{s}_{o2}, \cdots, \boldsymbol{s}_{o(\hat{N}-1)})$ denote the set of even and odd blocks respectively, where $\boldsymbol{s}_{ei} = \boldsymbol{x}_{[2(i-1)L+1:2iL]}$ for $i \in [\hat{N}]$ and $\boldsymbol{s}_{oi} = \boldsymbol{x}_{[(2i-1)L+1:(2i+1)L]}$ for $i \in [\hat{N}-1]$.

Since the k-substring edit is located in a specific interval with the length bounded by L, then it will occur in at least one block of either S_e and S_o. In the following, we apply a modified syndrome compression code [15] to correct a strict k-substring edit in a length-$2L$ string.

Based on Theorem 3, for $\boldsymbol{u} \in \Sigma_2^{2L}$ and a labeling function f over $B(\boldsymbol{u})$, the decoder can recover $\boldsymbol{u}$ by $\boldsymbol{v} \in D(\boldsymbol{u})$, a, and $f(\boldsymbol{u}) \bmod a$. Furthermore, a complex labeling function f does not affect the redundancy of the code since the redundancy $2\log a$ is affected by the size of $B(\boldsymbol{u})$. Since a strict k-substring edit can be viewed as a k-burst deletion followed by a k-burst insertion, we introduce a labeling function that can correct at most $2k$ insertions, deletions, and substitutions from [13].

Theorem 10 (cf. [13])**.** *Given a constant k, $t = 2k$, and $L = O((\log n)^2)$, There exists a labeling function $g : \Sigma_2^{2L} \to \Sigma_{2^{\mathcal{R}(t,2L)}}$ such that for any two distinct strings $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ confusable under at most t insertions, deletions, and substitutions,*

we have $g(\boldsymbol{s}_1) \neq g(\boldsymbol{s}_2)$*, where* $\mathcal{R}(t, 2L) = ((t^2+1)(2t^2+1)+ 2t^2(t-1))\log 2L + o(\log 2L) = O(\log\log n) + o(\log\log n)$.

Based on Lemma 2, given $\boldsymbol{u} \in \Sigma_2^{2l}$, the size of the confusable set $B(\boldsymbol{u})$ satisfies $\|B(\boldsymbol{u})\| < (2L+k)^2(k+1)^4 2^{2k}$. Then for each $\boldsymbol{s}_{ei} \in \Sigma_2^{2L}$ and $\boldsymbol{w}_{ei} \in B(s_{ei})$, there exists an integer a_{ei} such that $g(\boldsymbol{s}_{ei}) \not\equiv g(\boldsymbol{w}_{ei}) \bmod a_{ei}$ for $i \in [\hat{N}]$, where $a_{ei} \leq 2^{\log \|B(\boldsymbol{s}_{ei})\| + o(\log L)}$. The same property holds for each $\boldsymbol{s}_{oi} \in \Sigma_2^{2L}$ for $i \in [\hat{N}-1]$.

Based on the two sets of messages S_e, S_o, we have the following construction for a k-substring edit.

Construction 11. *Let* $\boldsymbol{\beta} = (\beta_1, \beta_2, \cdots, \beta_6)$*. Given* $\boldsymbol{x} \in \mathcal{C}_{loc}(\beta_1, \beta_2)$ *with length* n*, we generate two sets of message blocks* S_e *and* S_o*. Let* $\beta_3, \cdots, \beta_6 < \alpha$*. Then we have*

$$\mathcal{C}_{strict}(\alpha, \boldsymbol{\beta}) = \{\boldsymbol{x} \in \mathcal{C}_{loc}(\beta_1, \beta_2),$$
$$\sum_{i=1}^{\hat{N}} a_{ei} = \beta_3 \bmod \alpha, \sum_{i=1}^{\hat{N}} (g(\boldsymbol{s}_{ei}) \bmod a_{ei}) = \beta_4 \bmod \alpha,$$
$$\sum_{i=1}^{\hat{N}-1} a_{oi} = \beta_5 \bmod \alpha, \sum_{i=1}^{\hat{N}-1} (g(\boldsymbol{s}_{oi}) \bmod a_{oi}) = \beta_6 \bmod \alpha.\}$$

where α *satisfies* $\alpha \geq (2L+k)^2(k+1)^4 2^{2k} 2^{o(\log L)} > \max(a_{e1}, a_{o1} \cdots, a_{e(\hat{N}-1)}, a_{o(\hat{N}-1)}, a_{eN})$.

Note that a similar construction appeared recently in [10], [11] for burst deletions. However, our construction includes a more powerful error-locating code and modified modulus so that our code can correct a strict k-substring edit.

Lemma 12. *Given a constant* k*, the error-correcting code* $\mathcal{C}_{strict}(\boldsymbol{\beta}, \alpha)$ *in Construction 11 can correct a strict* k*-substring edit error with the redundancy of roughly* $\log n + 16\log\log n + o(\log\log n)$ *bits.*

B. Error-correcting code for a k-burst substitution

In this subsection, we present a code that can correct a k-burst substitution error.

Construction 13 (cf. [20], Fire code)**.** *Let* $g_0(x)$ *be an irreducible polynomial of degree* $m \geq k$ *that does not divide* $x^{2k-1} - 1$*. Then, there exists a linear cyclic code (called Fire code) of length* $n_1 = \mathrm{LCM}(2k-1, 2^m - 1)$ *with the* generator polynomial $g(x) = (x^{2k-1} - 1)g_0(x)$ *and* $\deg(g(x)) = m + 2k - 1$*. Then the Fire code* $\mathcal{C}_F$ *can be represented as* $[n_1, n]$ *code with the codeword length* n_1 *and the dimension* $n = n_1 - (m + 2k - 1)$.

Theorem 14 (cf. [20])**.** *The Fire code* $\mathcal{C}_F$ *can correct a single* k*-burst substitution.*

Then the redundancy of the Fire code can be presented by the following lemma.

Lemma 15. *Given a constant* k*, the Fire code* $\mathcal{C}_F$ *corrects a* k*-burst substitution with the redundancy roughly* $\log n + o(\log n)$.

Hence, given $\boldsymbol{x} \in \Sigma_2^n$, there exists a function $h_F(\boldsymbol{x})$ of length roughly $\log n + o(\log n)$ such that $(\boldsymbol{x}, h_F(\boldsymbol{x})) \in \mathcal{C}_F$ is capable of fighting against a k-burst substitution.

C. Combined error-correcting codes

Based on Lemma 5, given $\boldsymbol{x} \in \mathcal{D}_{\mathcal{P},\delta}(n) \cap \Sigma_2^n$, the receiver can correct a k-substring edit from $\boldsymbol{y} \in D(\boldsymbol{x})$ if $h_F(\boldsymbol{x})$ and $(\boldsymbol{\beta}, \alpha)$ are sent to the receiver by an error-free channel. For simplicity, let $\boldsymbol{r_x} := (h_F(\boldsymbol{x}), \boldsymbol{\beta}, \alpha)$ be a binary representation of the data. Then each codeword of the final error-correcting codes can be generated by concatenating two parts, i.e., $(\boldsymbol{x}, \boldsymbol{r_x})$. Furthermore, we may add a buffer between $\boldsymbol{x}$ and $\boldsymbol{r_x}$ such that a k-substring edit affects either $\boldsymbol{x}$ or $\boldsymbol{r_x}$. Finally, We also need another function that can detect or correct a k-substring edit occurring in $\boldsymbol{r_x}$. We start by finding a suitable buffer $\boldsymbol{b_x}$.

Since a k-substring edit should not affect both $\boldsymbol{x}$ and $\boldsymbol{r_x}$, the length of the buffer satisfies $|\boldsymbol{b_x}| > k$. The buffer in the following lemma helps distinguish whether the strict k-substring edit affects $\boldsymbol{x}$ or $\boldsymbol{r_x}$.

Lemma 16. *Given a string* $\boldsymbol{w} = \boldsymbol{x}\boldsymbol{b_x}\boldsymbol{u}$ *with* $\boldsymbol{x} \in \Sigma_2^n$ *and* $\boldsymbol{u} \in \Sigma_2^*$*. A buffer* $\boldsymbol{b_x} = 1^{k+1}0^{k+1}1^{k+1}$ *can distinguish whether a strict* k*-substring edit affect either* $\boldsymbol{x}$ *or* $\boldsymbol{u}$*.*

Then a burst-substitution-detecting function $\mathcal{E}_1$ suffices to decode $\boldsymbol{x}$ if it can detect an k-burst substitution in $\boldsymbol{r_x}$.

Lemma 17. *Given a constant* k *and* $\boldsymbol{b_x} = 1^{k+1}0^{k+1}1^{k+1}$*, a burst-substitution-detecting function* $\mathcal{E}_1$ *in* $\boldsymbol{x}\boldsymbol{b_x}\boldsymbol{r_x}\mathcal{E}_1(\boldsymbol{r_x})$ *is sufficient to decode* $\boldsymbol{x}$ *for a* k*-substring edit.*

The simplest burst-substitution-detecting function is a parity-checking function. Given $\boldsymbol{r_x}$, let $L_1 = |\boldsymbol{r_x}| > k$. Then we partition $\boldsymbol{r_x}$ into $T = \lceil L_1/(k+1) \rceil$ blocks of length $(k+1)$, i.e., $\boldsymbol{w}_j$ for $j \in [T]$, where additional 0s are appended if the last block has less than $(k+1)$ binary symbols. Then the error-detecting function $\mathcal{E}_1 : \Sigma_2^{L_1} \to \Sigma_2^{2k+2}$ appends $\boldsymbol{\gamma}_1$ and $\boldsymbol{\gamma}_2$ in the binary form (each with length $(k+1)$) following $\boldsymbol{r_x}$, where

$$\boldsymbol{\gamma}_1 = \left(\sum_{j=1}^{\lceil T/2 \rceil} \boldsymbol{w}_{2j-1}\right) \bmod 2^{k+1}, \boldsymbol{\gamma}_2 = \left(\sum_{j=1}^{\lfloor T/2 \rfloor} \boldsymbol{w}_{2j}\right) \bmod 2^{k+1}.$$

The final construction is shown below.

Construction 18. *Given a constant* k*,* $\boldsymbol{b}_x = 1^{k+1}0^{k+1}1^{k+1}$*, we have a construction* $\mathcal{C}_N$ *as*

$$\mathcal{C}_N = \{\boldsymbol{x}\boldsymbol{b_x}\boldsymbol{r_x}\boldsymbol{\gamma}_1\boldsymbol{\gamma}_2 \in \Sigma_2^N, \boldsymbol{x} \in \Sigma_2^n \cap \mathcal{D}_{\mathcal{P},\delta}(n)\},$$

where $\boldsymbol{r_x} = (h_F(\boldsymbol{x}), \boldsymbol{\beta}, \alpha)$ *and* $\boldsymbol{\gamma}_1\boldsymbol{\gamma}_2$ *are in binary form generated by* $\mathcal{E}_1(\boldsymbol{r_x})$ *in each codeword* $\boldsymbol{x}\boldsymbol{b_x}\boldsymbol{r_x}\boldsymbol{\gamma}_1\boldsymbol{\gamma}_2 \in \Sigma_2^N$.

Theorem 19. *The error-correcting code* $\mathcal{C}_N$ *in Construction 18 can correct a* k*-substring edit with the redundancy of roughly* $2\log N + o(\log N)$ *bits, where* $N = n + 2\log n + o(\log n)$.

D. Time complexity

Given a constant k, the encoding process consists of encoding message to a $(\mathcal{P}, \delta)$-dense string [11], appending a buffer, generating $\boldsymbol{r_x}$, and appending $\boldsymbol{\gamma}_1$ and $\boldsymbol{\gamma}_2$ with time complexity polynomial with respect to N. Similarly, the time complexity of the decoder is also polynomial with respect to N.

REFERENCES

[1] P. Fire, *A class of multiple-error-correcting binary codes for non-independent errors*. Stanford University, 1959, vol. 55.

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

[3] W. Zhou, S. Lin, and K. Abdel-Ghaffar, "Burst or random error correction based on Fire and BCH codes," in *2014 Information Theory and Applications Workshop (ITA)*. IEEE, 2014, pp. 1–5.

[4] L. Cheng, T. G. Swart, H. C. Ferreira, and K. A. Abdel-Ghaffar, "Codes for correcting three or more adjacent deletions or insertions," in *2014 IEEE International Symposium on Information Theory*. IEEE, 2014, pp. 1246–1250.

[5] C. Schoeny, A. Wachter-Zeh, R. Gabrys, and E. Yaakobi, "Codes correcting a burst of deletions or insertions," *IEEE Transactions on Information Theory*, vol. 63, no. 4, pp. 1971–1985, 2017.

[6] C. Schoeny, F. Sala, and L. Dolecek, "Novel combinatorial coding results for DNA sequencing and data storage," in *2017 51st Asilomar Conference on Signals, Systems, and Computers*. IEEE, 2017, pp. 511–515.

[7] A. Lenz and N. Polyanskii, "Optimal codes correcting a burst of deletions of variable length," in *2020 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2020, pp. 757–762.

[8] R. Bitar, S. K. Hanna, N. Polyanskii, and I. Vorobyev, "Optimal codes correcting localized deletions," in *2021 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2021, pp. 1991–1996.

[9] S. K. Hanna and S. El Rouayheb, "Codes for correcting localized deletions," *IEEE Transactions on Information Theory*, vol. 67, no. 4, pp. 2206–2216, 2021.

[10] S. Wang, Y. Tang, R. Gabrys, and F. Farnoud, "Permutation codes for correcting a burst of at most t deletions," in *2022 58th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2022, pp. 1–6.

[11] S. Wang, Y. Tang, J. Sima, R. Gabrys, and F. Farnoud, "Non-binary codes for correcting a burst of at most t deletions," 2022. [Online]. Available: https://arxiv.org/abs/2210.11818

[12] J. Sima and J. Bruck, "On optimal k-deletion correcting codes," *IEEE Transactions on Information Theory*, vol. 67, no. 6, pp. 3360–3375, 2020.

[13] J. Sima, R. Gabrys, and J. Bruck, "Optimal systematic t-deletion correcting codes," in *2020 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2020, pp. 769–774.

[14] Y. Tang, H. Lou, and F. Farnoud, "Error-correcting codes for short tandem duplications and at most p substitutions," in *2021 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2021, pp. 1835–1840.

[15] J. Sima, R. Gabrys, and J. Bruck, "Syndrome compression for optimal redundancy codes," in *2020 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2020, pp. 751–756.

[16] D. Deamer, M. Akeson, and D. Branton, "Three decades of nanopore sequencing," *Nature Biotechnology*, vol. 34, no. 5, pp. 518–524, May 2016.

[17] Y. Li, S. Wang, C. Bi, Z. Qiu, M. Li, and X. Gao, "Deepsimulator1. 5: a more powerful, quicker and lighter simulator for nanopore sequencing," *Bioinformatics*, vol. 36, no. 8, pp. 2578–2580, 2020.

[18] A. Orlitsky, "Interactive communication: Balanced distributions, correlated files, and average-case complexity," in *[1991] Proceedings 32nd Annual Symposium of Foundations of Computer Science*, Oct. 1991, pp. 228–238.

[19] C. R. Mehta and N. R. Patel, "IBM SPSS exact tests," *Armonk, NY: IBM Corporation*, pp. 23–24, 2011.

[20] R. E. Blahut, *Algebraic codes for data transmission*. Cambridge university press, 2003.