Asymptotically Optimal Codes Correcting One Substring Edit

Yuting Li*, Yuanyuan Tang†, Hao Lou†, Ryan Gabrys‡, and Farzad Farnoud*†
*Computer Science, University of Virginia, USA, mzy8rp@virginia.edu
†Electrical & Computer Engineering, University of Virginia, USA,
{yt5tz,hl2nu,farzad}@virginia.edu
‡Calit2, University of California-San Diego, USA, rgabrys@ucsd.edu

***Abstract*—The substring edit error is the operation of replacing a substring $\boldsymbol{u}$ of $\boldsymbol{x}$ with another string $\boldsymbol{v}$, where the lengths of $\boldsymbol{u}$ and $\boldsymbol{v}$ are bounded by a given constant k. It encompasses localized insertions, deletions, and substitutions within a window. Codes correcting one substring edit have redundancy at least $\log n + k$. In this paper, we construct codes correcting one substring edit with redundancy $\log n + O(\log\log n)$, which is asymptotically optimal. The full version of this paper is available online.**[1]

I. Introduction

In data transmission and storage, especially in data storage in DNA [1] and other emerging media, the need for robust error correction for a diverse set of errors is an important and challenging problem. This paper presents a family of asymptotically optimal codes capable of correcting a burst of localized edits, encompassing combinations of insertions, deletions, and substitutions. In addition to their application in error-correction, the codes provide a promising approach to the challenge of synchronizing multiple copies of related files, where bursts of edits are common. The data synchronization problem, also known as document exchange, has been explored in various works demonstrating that error-correcting codes, especially those accommodating file edits, can substantially reduce the communication required to maintain file consistency [2].

Specifically, we focus on correcting a single burst of errors of length at most k, which we refer to as a k-***substring edit***. Such an error may be the result of insertions, deletions, or substitutions, occurring within a bounded interval of our strings. Formally, a k-substring edit in a string $\boldsymbol{x}$ is the operation of replacing a substring $\boldsymbol{u}$ of $\boldsymbol{x}$ with another string $\boldsymbol{v}$, where $|\boldsymbol{u}|, |\boldsymbol{v}| \leq k$. An analysis of real and simulated data in [3] reveals that substring edits are common in file synchronization and DNA storage, where viewing errors as substring edits leads to smaller redundancies.

In this paper, we assume that k is a fixed constant. Notice that for the setting where we replace a string $\boldsymbol{u}$ with another string $\boldsymbol{v}$ of the same length (i.e., $|\boldsymbol{u}| = |\boldsymbol{v}|$), a substring edit is equivalent to a burst of substitutions. Furthermore, if $\boldsymbol{v}$ is the empty string, then a substring edit is equivalent to a burst of deletions. Analogously, if $\boldsymbol{u}$ is the empty string, then the substring edit is a burst of insertions. Despite the fact that these specialized types of substring edit have each received significant attention in the past, the problem of constructing codes for substring edits has received relatively less attention, with the exception of [3] and [4]. A critical observation in this context is that the ability to correct an arbitrarily long burst of deletions does not enable correcting a k-substring edit [3, Lemma 1].

Using simple counting arguments, it can be shown that a code correcting one substring edit has redundancy at least $\log n + k$ and at most roughly $2\log n$. In [3], the authors present an efficient construction for a code that corrects one substring edit with redundancy roughly $2\log n$, which matches the existential upper bound of redundancy. However, the question of whether it is possible to construct a code with less than $2\log n$ bits of redundancy remained open. In this paper, we provide an affirmative answer to this question and construct codes correcting one substring edit with redundancy roughly $\log n + O(\log\log n)$, i.e., at most $O(\log\log n)$ bits larger than the optimal redundancy. Hence, the codes we propose are asymptotically optimal in terms of redundancy and improve the state-of-the-art redundancy by a factor of 2.

The rest of the paper is organized as follows: In Section II, we introduce basic definitions and notation, review our techniques, and compare with related works. In Section III, we construct asymptotically optimal codes correcting one substring edit.

II. Notation, Overview, and Related Work

A. Basic Definitions and Notations

We will deal with two types of strings, strings over $\{0,1\}$ and strings over the set of nonnegative integers. When necessary to identify the alphabet, we refer to a string as a *binary string* or an *integer string*, respectively. Strings are denoted by bold symbols, e.g., $\boldsymbol{u} = u_1 \cdots u_n$. Let $[i,j]$ represent the integers $i, i+1, \ldots, j$. Furthermore, the set $[1,j]$ is denoted as $[j]$. The substring of $\boldsymbol{u}$ starting at position i and ending at position j is denoted by $\boldsymbol{u}_{[i,j]}$. We use $|\boldsymbol{u}|$ to denote the length of $\boldsymbol{u}$. For $\boldsymbol{z} \in \mathbb{N}^*$ and a positive integer A, we say that $\boldsymbol{z}$ is A-bounded if each symbol of $\boldsymbol{z}$ is at most A.

For binary strings $\boldsymbol{x}, \boldsymbol{y} \in \{0,1\}^*$, we denote a k-substring edit by $\boldsymbol{x} \to \boldsymbol{y}$. If $\boldsymbol{x} \to \boldsymbol{y}$ and $\boldsymbol{x} \neq \boldsymbol{y}$, then we may also write $\boldsymbol{x} \xrightarrow{\neq} \boldsymbol{y}$. In our construction, substring edits over integer strings also arise. For clarity, we use a different notation for

[1] https://www.ece.virginia.edu/~ffh8x/papers/eccss.pdf

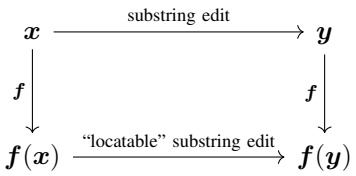

Figure 1: converting a substring edit to a "locatable" substring edit

these: For two integer strings $\boldsymbol{z}, \boldsymbol{w} \in \mathbb{N}^*$, we let $\boldsymbol{z} \Rightarrow \boldsymbol{w}$ denote a substring edit (not necessarily a k-substring edit) and let $\boldsymbol{z} \overset{j:a,b}{\Longrightarrow} \boldsymbol{w}$ denote the substring edit that deletes a substring of length a starting in position j, and inserts a substring of length b in its place. This operation is a $\max(a, b)$-substring edit.

A partition $\mathcal{P}$ of $\boldsymbol{x}$ is a rule under which one divides $\boldsymbol{x}$ into parts. We use $n_{\mathcal{P}}(\boldsymbol{x})$ to denote the number of parts and $\boldsymbol{x}^{\mathcal{P}}$ to denote the vector $(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{n_{\mathcal{P}}(\boldsymbol{x})})$, where $\boldsymbol{x}_i$, $1 \leq i \leq n_{\mathcal{P}}(\boldsymbol{x})$, denotes a part. We call a string $(\mathcal{P}, \delta)$-dense if all elements of $\boldsymbol{x}^{\mathcal{P}}$ have length at most δ. Suppose f is a function on strings. Then we use $\boldsymbol{f}^{\mathcal{P}}(\boldsymbol{x})$ to denote the string $f(\boldsymbol{x}_1)f(\boldsymbol{x}_2)\cdots f(\boldsymbol{x}_{n_{\mathcal{P}}(\boldsymbol{x})})$. When the partition $\mathcal{P}$ is clear from the context, we may simply use $\boldsymbol{f}(\boldsymbol{x})$ to denote $\boldsymbol{f}^{\mathcal{P}}(\boldsymbol{x})$. (Note that $f(\boldsymbol{x})$, in contrast to $\boldsymbol{f}(\boldsymbol{x})$, is simply the result of applying f to $\boldsymbol{x}$.) For a (binary or integer) string $\boldsymbol{z}$, its VT sketch is defined as $\mathrm{VT}(\boldsymbol{z}) = \sum_{i=1}^{|\boldsymbol{z}|} i z_i$.

B. Overview of the Techniques

The overview of our approach to correcting one substring edit error is given in Figure 1. Here, $\boldsymbol{x}$ is the input to the channel while $\boldsymbol{y}$ is the output. The first step in correcting the error is to approximately identify its position. To achieve this, we rely on a key insight that goes back to the pioneering work of Levenshtein [5] on correcting a burst of at most 2 deletions. Namely, while the VT sketch applied to $\boldsymbol{x}$ can locate only a single deletion, it can be more powerful when applied to carefully designed functions of $\boldsymbol{x}$. Previous work has used this observation to correct deletions within a window, as well as to correct one deletion or one adjacent transposition[6][7][8]. Our main contribution in this direction is to introduce a novel mapping that, when used in conjunction with a VT sketch, can locate the position of any k-substring edit to within a small interval. In particular, this mapping allows us to locate a burst of substitutions, a challenging task since such an edit may leave the sequence composed of lengths between "markers" untouched.

To construct the aforementioned mapping, denoted in Figure 1 by $\boldsymbol{f}$, we identify a set of conditions on a substring edit over an integer string that, if satisfied, the location of the substring edit can be approximately identified using the VT sketch and some other small amount of information (Lemma 2). Such a substring edit is called "locatable", a term that is defined precisely in Definition 1. Specifically, our approach will be to find a partition rule $\mathcal{P}$ and a function f such that if $\boldsymbol{x} \overset{\neq}{\Longrightarrow} \boldsymbol{y}$, then $\boldsymbol{f}(\boldsymbol{y}) = \boldsymbol{f}^{\mathcal{P}}(\boldsymbol{y})$ is obtained from $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{f}^{\mathcal{P}}(\boldsymbol{x})$ by a locatable substring edit. Then we restrict $\boldsymbol{x}$ to be $(\mathcal{P}, \delta)$-dense. If $\boldsymbol{x} \overset{\neq}{\Longrightarrow} \boldsymbol{y}$, we treat $\boldsymbol{f}^{\mathcal{P}}(\boldsymbol{x})$ and $\boldsymbol{f}^{\mathcal{P}}(\boldsymbol{y})$ as the $\boldsymbol{z}$ and $\boldsymbol{w}$ in Lemma 2, and determine the edit position of this locatable substring edit to within an interval of length $O(\delta) = O(\log n)$. Since we restrict $\boldsymbol{x}$ to be $(\mathcal{P}, \delta)$-dense, which means $|\boldsymbol{x}_i| \leq \delta$ for any i, we can determine the original edit position to within an interval of length $O(\delta^2)$ (a technique used in [7]), which is then corrected. The case in which $\boldsymbol{x} = \boldsymbol{y}$ will also be handled in Theorem 11. The VT sketch used in our construction will require a redundancy of about $\log n$ bits, while the other components will have negligible redundancy.

C. Related Works

Codes correcting localized errors have been a subject of extensive study over time. Fire codes [9] are designed to construct a burst of k substitutions with roughly $\log n$ redundancy. Several works, including [4], [10], [11], have construct burst-deletion correcting codes, where the length of the bursts is known in advance. More related to this work are codes that can correct a variable number of deletions. In a pioneering work by Levenshtein's work [5], optimal codes are constructed to correct a burst of at most two consecutive deletions.

For longer deletions and non-consecutive but localazide deletions, several works [3], [6]–[8] aim to first approximately locate the deletion. Their method uses the VT sketch of an integer string that is related to the original binary string to locate the approximate deletion position. Specifically, in [6], [7], and [3], each symbol of the integer string is the length of each part of the original binary string with respect to some partition. Using this method, Lenz et al. [6] propose codes correcting a variable-length burst of deletions with nearly optimal redundancy, but the deletions still need to be consecutive. Bitar et al. [7] later construct nearly optimal codes that remove the restriction of consecutive deletions, so that it could correct deletions within a window that do not necessarily occur consecutively. Also using VT sketch of the length integer string, Tang et al. [3] construct codes correcting a substring edit, but the VT sketch of the length integer string can locate the edit position only when the substring edit changes the length of the original binary string. So the construction from [3] needs another $\log n$ bits to handle the other cases, and the total redundancy is about $2 \log n$ bits, which is not optimal. By using a different integer string, Gabrys et al. [8] further construct codes correcting one deletion or one adjacent transposition with near-optimal redundancy, but their approach does not address the setup where additional edits may occur.

III. Asymptotically Optimal Codes Correcting One Substring Edit

In this section, we present the code construction, prove its correctness, and determine its redundancy. Based on our earlier discussion in Subsection II-B, the construction has three interrelated components: a) identifying a set of conditions that make a substring edit over an integer string "locatable" and showing that the position of a locatable substring edit can be found approximately; b) constructing a partition rule $\mathcal{P}$ and a mapping f such that if $\boldsymbol{x} \overset{\neq}{\Longrightarrow} \boldsymbol{y}$, then $\boldsymbol{f}^{\mathcal{P}}(\boldsymbol{y})$ is

obtained from $\boldsymbol{f}^{\mathcal{P}}(\boldsymbol{x})$ through a "locatable" substring edit; and c) correcting the edit in $\boldsymbol{y}$ and recovering $\boldsymbol{x}$. We discuss each in a subsection below. In the next subsection, we discuss each of these components. Then we prove the correctness of the overall approach.

A. Components of the Code Construction

a) Locatable substring edits: We will now define the notion of locatable substring edit in Definition 1, and prove Lemma 2, which roughly says that for two δ-bounded integer strings $\boldsymbol{z}$ and $\boldsymbol{w}$, if $\boldsymbol{w}$ is obtained from $\boldsymbol{z}$ by a locatable substring edit at position j, then given $\boldsymbol{w}$, $\mathrm{VT}(\boldsymbol{z})$, and some other little information, one can determine j to within an interval of length $O(\delta)$.

Definition 1. *For two positive integer strings $\boldsymbol{z}$ and $\boldsymbol{w}$, we say that $\boldsymbol{w}$ is obtained from $\boldsymbol{z}$ by a* **locatable** *K-substring edit at position j if $\boldsymbol{z} \xrightarrow{j:a,b} \boldsymbol{w}$, where $a, b \leq K$, and*

1) *The difference of the sum of $\boldsymbol{z}$ and the sum of $\boldsymbol{w}$ is bounded by all w_i, namely $\left|\sum_{j=1}^{|\boldsymbol{z}|} z_j - \sum_{j=1}^{|\boldsymbol{w}|} w_j\right| < w_i$ for all i's.*
2) *$|\boldsymbol{z}| \neq |\boldsymbol{w}|$ or $\sum_{j=1}^{|\boldsymbol{z}|} z_j \neq \sum_{j=1}^{|\boldsymbol{w}|} w_j$.*

Now we show that for A-bounded strings $\boldsymbol{z}$ and $\boldsymbol{w}$, if $\boldsymbol{w}$ is obtained from $\boldsymbol{z}$ by a locatable K-substring edit at position j, then one can determine j to within an interval of length $O(A)$ given $\mathrm{VT}(\boldsymbol{z}) \bmod O(m)$ (for some parameter m specified in the lemma statement) and some other little information of $\boldsymbol{z}$.

Lemma 2. *For A-bounded strings $\boldsymbol{z}$ and $\boldsymbol{w}$, if $\boldsymbol{w}$ is obtained from $\boldsymbol{z}$ by a locatable K-substring edit at position j, then one can determine j to within an interval of length $6K^2A$ given $\boldsymbol{w}$, $|\boldsymbol{z}|-|\boldsymbol{w}|$, $\sum_{i=1}^{|\boldsymbol{z}|} z_i - \sum_{i=1}^{|\boldsymbol{w}|} w_i$, and $\mathrm{VT}(\boldsymbol{z}) \bmod m$, where $m > 8K^2\left(\sum_{i=1}^{|\boldsymbol{z}|} z_i + A\right)$.*

Proof. Suppose $\boldsymbol{z} \xrightarrow{j:a,b} \boldsymbol{w}$, where $a, b \leq K$ and $j \in [1, |\boldsymbol{z}| - a + 1]$. Let $\eta(v, \boldsymbol{z}, \boldsymbol{w}) := (|\boldsymbol{z}| - |\boldsymbol{w}|)\sum_{i=v}^{|\boldsymbol{w}|} w_i + (\sum_{i=1}^{|\boldsymbol{z}|} z_i - \sum_{i=1}^{|\boldsymbol{w}|} w_i)v$, where v is an integer. Note that since we know $\boldsymbol{w}$, $|\boldsymbol{z}|-|\boldsymbol{w}|$, and $\sum_{i=1}^{|\boldsymbol{z}|} z_i - \sum_{i=1}^{|\boldsymbol{w}|} w_i$, we know the expression of $\eta(\cdot, \boldsymbol{z}, \boldsymbol{w})$. Let $\Delta\mathrm{VT} := \mathrm{VT}(\boldsymbol{z}) - \mathrm{VT}(\boldsymbol{w})$. We will prove the following.

1) $\eta(\cdot, \boldsymbol{z}, \boldsymbol{w})$ is strictly monotone on $[1, |\boldsymbol{w}| + 1]$.
2) $|\Delta\mathrm{VT} - \eta(j, \boldsymbol{z}, \boldsymbol{w})| < 3K^2A$.
3) $|\Delta\mathrm{VT}| \leq 4K^2\left(\sum_{i=1}^{|\boldsymbol{z}|} z_i + A\right)$.

For 1), note that

$$\eta(v, \boldsymbol{z}, \boldsymbol{w}) - \eta(v-1, \boldsymbol{z}, \boldsymbol{w}) = (|\boldsymbol{z}| - |\boldsymbol{w}|)w_v + \sum_{i=1}^{|\boldsymbol{z}|} z_i - \sum_{i=1}^{|\boldsymbol{w}|} w_i.$$

If $|\boldsymbol{z}| \neq |\boldsymbol{w}|$, since $\boldsymbol{w}$ is obtained from $\boldsymbol{z}$ by a locatable K-substring edit, we have $\left|\sum_{i=1}^{|\boldsymbol{z}|} z_i - \sum_{i=1}^{|\boldsymbol{w}|} w_i\right| < w_v$. So $\eta(\cdot, \boldsymbol{z}, \boldsymbol{w})$ is strictly monotone on $[1, |\boldsymbol{w}| + 1]$. If $|\boldsymbol{z}| = |\boldsymbol{w}|$, since $\boldsymbol{w}$ is obtained from $\boldsymbol{z}$ by a locatable K-substring edit, we have $\left|\sum_{i=1}^{|\boldsymbol{z}|} z_i - \sum_{i=1}^{|\boldsymbol{w}|} w_i\right| \neq 0$. So, $\eta(\cdot, \boldsymbol{z}, \boldsymbol{w})$ is strictly monotone on $[1, |\boldsymbol{w}| + 1]$.

For 2), since $\boldsymbol{z} \xrightarrow{j:a,b} \boldsymbol{w}$, we have

$$\Delta\mathrm{VT} = f(j, \boldsymbol{z}, \boldsymbol{w}) + \sum_{i=1}^{a-1} i z_{j+i} - \sum_{i=1}^{b-1} i w_{j+i} - (|\boldsymbol{z}| - |\boldsymbol{w}|)\sum_{i=j}^{j+b-1} w_i.$$

Since $\boldsymbol{z}$ and $\boldsymbol{w}$ are A-bounded strings, we have $\left|\sum_{i=1}^{a-1} i z_{j+i}\right| \leq a^2A$, $\left|\sum_{i=1}^{b-1} i w_{j+i}\right| \leq b^2A$, and $\left|\sum_{i=j}^{j+b-1} w_i\right| \leq bA$. So we have

$$\begin{aligned} |\eta(j, \boldsymbol{z}, \boldsymbol{w}) - \Delta\mathrm{VT}| &\leq (a^2 + b^2)A + |a-b|bA \\ &< 3K^2A. \end{aligned} \tag{1}$$

For 3), since $\eta(\cdot, \boldsymbol{z}, \boldsymbol{w})$ is monotone on $[1, |\boldsymbol{w}| + 1]$,

$$\begin{aligned} |\eta(j, \boldsymbol{z}, \boldsymbol{w})| &\leq |\eta(1, \boldsymbol{z}, \boldsymbol{w})| + |\eta(|\boldsymbol{w}| + 1, \boldsymbol{z}, \boldsymbol{w})| \\ &\leq K^2\left(\sum_{i=1}^{|\boldsymbol{z}|} z_i + A\right). \end{aligned} \tag{2}$$

Thus, by (1) and (2), we have

$$\begin{aligned} |\Delta\mathrm{VT}| &\leq |\Delta\mathrm{VT} - \eta(j, \boldsymbol{z}, \boldsymbol{w})| + |\eta(j, \boldsymbol{z}, \boldsymbol{w})| \\ &\leq 4K^2\left(\sum_{i=1}^{|\boldsymbol{z}|} z_i + A\right). \end{aligned}$$

Now since we know $\mathrm{VT}(\boldsymbol{z}) \bmod m$, we can find $\Delta\mathrm{VT}$. Since $\eta(\cdot, \boldsymbol{z}, \boldsymbol{w})$ is strictly monotone on $[1, |\boldsymbol{w}| + 1]$ and $|\Delta\mathrm{VT} - \eta(j, \boldsymbol{z}, \boldsymbol{w})| < 3K^2A$, we can determine j to within an interval of length $6K^2A$. □

b) Constructing a mapping to convert a general substring edit to a locatable edit: Our goal here is to construct a partition $\mathcal{P}$ and a function f over strings such that if $\boldsymbol{x} \xrightarrow{\neq} \boldsymbol{y}$, then $\boldsymbol{f}(\boldsymbol{y}) = \boldsymbol{f}^{\mathcal{P}}(\boldsymbol{y})$ is obtained from $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{f}^{\mathcal{P}}(\boldsymbol{x})$ by a locatable substring edit. We first give the partition rule $\mathcal{P}$. Let $\boldsymbol{p} = 0^k1^k$, which we call a pattern.

We define the partition rule $\mathcal{P}$ as follows. For $\boldsymbol{x} \in \{0,1\}^*$, let $\boldsymbol{x}^{\mathcal{P}} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{n_{\mathcal{P}}(\boldsymbol{x})})$, where the starting position of $\boldsymbol{x}_1$ is the leftmost bit of $\boldsymbol{x}$ and the starting position of $\boldsymbol{x}_i$ ($i \geq 2$) is the leftmost bit of the $(i-1)$th $\boldsymbol{p}$ in $\boldsymbol{x}$.

Example 3. *Let $k = 3$, $\boldsymbol{p} = 0^31^3$, and*

$$\boldsymbol{x} = \overbrace{01001}^{\boldsymbol{x}_1}\overbrace{\underbrace{000111}_{\boldsymbol{p}}100}^{\boldsymbol{x}_2}\overbrace{\underbrace{000111}_{\boldsymbol{p}}101001}^{\boldsymbol{x}_3}\overbrace{\underbrace{000111}_{\boldsymbol{p}}1}^{\boldsymbol{x}_4}.$$

To get $\boldsymbol{x}^{\mathcal{P}}$, we first find all the patterns, and break it up at the left side of each pattern. So we get $\boldsymbol{x}^{\mathcal{P}} = (01001, 000111100, 000111101001, 0001111)$.

We now explain the main idea behind the construction of the function f. For any lowercase function u over strings, we

use the uppercase function U to denote the sum of all the symbols of $\boldsymbol{u}(\boldsymbol{x})$, namely

$$U(\boldsymbol{x}) := \sum_{i=1}^{n_{\mathcal{P}}(\boldsymbol{x})} u(\boldsymbol{x}_i) = \sum_{i=1}^{|\boldsymbol{u}(\boldsymbol{x})|} u(\boldsymbol{x}_i),$$

and use ΔU to denote $U(\boldsymbol{x}) - U(\boldsymbol{y})$. For simplicity, we only consider the second condition of the locatable substring edit (see the full version for a complete treatment). Applied to the substring edit $f(\boldsymbol{x}) \Rightarrow f(\boldsymbol{y})$, the second condition of Definition 1 requires that if $\boldsymbol{x} \xrightarrow{\neq} \boldsymbol{y}$, then $|\boldsymbol{f}(\boldsymbol{x})| \neq |\boldsymbol{f}(\boldsymbol{y})|$ or $\Delta F \neq 0$. Note that if the number of patterns changes ($n_{\mathcal{P}}(\boldsymbol{x}) \neq n_{\mathcal{P}}(\boldsymbol{y})$), then $|\boldsymbol{f}(\boldsymbol{x})| \neq |\boldsymbol{f}(\boldsymbol{y})|$. Hence, we assume that the number of patterns does not change. We want to ensure that $\Delta F = F(\boldsymbol{x}) - F(\boldsymbol{y}) \neq 0$. To better explain our approach, we first outline two different approaches that have increasing levels of complexity before arriving at our actual solution, which is described as the third attempt.

First attempt. For a string $\boldsymbol{x}$, let $l(\boldsymbol{x})$ represent the length of $\boldsymbol{x}$. Note that $L(\boldsymbol{x})$ is also the length of $\boldsymbol{x}$. In our first attempt, we let $f(\cdot) = l(\cdot)$. Then $F(\boldsymbol{x}) = L(\boldsymbol{x})$. In this case, if the lengths of the deleted and inserted strings differ, then $\Delta F \neq 0$. However, $\Delta F = 0$ occurs when the substring edit is a burst of substitutions. In fact, in [3], the authors use $f(\cdot) = l(\cdot)$, and use another $\log n$ bits of redundancy to handle a burst of substitutions separately. This leads to a redundancy of $2 \log n$ bits. In this work, we will introduce two other novel functions, d (second attempt), and n_{1^k} (third attempt), to resolve this issue.

Second attempt. Let $d : \{0,1\}^n \to [2^k]$ be a function that can detect a burst of k substitutions, where the jth bit of $d(\boldsymbol{x})$ is the parity of every kth index of $\boldsymbol{x}$ stating at position j, namely,

$$d(\boldsymbol{x}) := \left(\bigoplus_{i \equiv_k 1} x_i, \bigoplus_{i \equiv_k 2} x_i, \ldots, \bigoplus_{i \equiv_k k} x_i \right),$$

where $\oplus$ is addition in $\mathbb{F}_2$. Note that $|\Delta D| \leq 2^k$. We let $f(\cdot) = d(\cdot) + Cl(\cdot)$, where C is a sufficiently large constant. Then, $\Delta F = \Delta D + C\Delta L$.

Now, if $\Delta L \neq 0$, then $\Delta F \neq 0$ provided that we choose $C > 2^k$. If $\Delta L = 0$, then the edit transforming $\boldsymbol{x}$ into $\boldsymbol{y}$ is a burst of substitutions. At first glance, it may seem that in this case $\Delta D \neq 0$, and thus $\Delta F \neq 0$. But while this is the case if this burst of substitutions does not involve any patterns, it may not be the case otherwise. The next example shows that if this burst of substitutions involves some patterns, then ΔD may be 0.

Example 4. *Let $k = 3$, $\boldsymbol{p} = 0^3 1^3$, and*

$$\boldsymbol{x} = \overbrace{000011}^{\boldsymbol{x}_1} \overbrace{\underbrace{000111}_{\boldsymbol{p}}}^{\boldsymbol{x}_2} 001111,$$

$$\boldsymbol{y} = \overbrace{00001100001}^{\boldsymbol{y}_1} \overbrace{\underbrace{000111}_{\boldsymbol{p}}}^{\boldsymbol{y}_2} 1,$$

where the edited substring is in bold and red. Then $\boldsymbol{l}(\boldsymbol{x}) = l(\boldsymbol{x}_1)l(\boldsymbol{x}_2) = (6, 12)$, $\boldsymbol{l}(\boldsymbol{y}) = (11, 7)$. $\boldsymbol{d}(\boldsymbol{x}) = (011, 001) = (3, 1)$, $\boldsymbol{d}(\boldsymbol{y}) = (001, 011) = (1, 3)$. Then $\Delta L = 0$ and $\Delta D = 0$, so $\Delta F = 0$. Thus, $\boldsymbol{f}(\boldsymbol{x}) \Rightarrow \boldsymbol{f}(\boldsymbol{y})$ is not a locatable substring edit.

Third attempt. Recall that the method in the second attempt may not work only if the substring edit is a burst of substitutions that involves some patterns. By careful argument, one can find that the only case where $\Delta F = 0$ in the second attempt is when one pattern $0^k 1^k$ is shifted because of this burst of substitutions. So, we need a function, say g, such that $\Delta G \neq 0$ in this case, and thus our f will be of the form $f(\cdot) = d(\cdot) + Bg(\cdot) + Cl(\cdot)$, where B is a sufficiently large constant and C is much larger than B. This way, if $\Delta L \neq 0$, then $\Delta F \neq 0$. If $\Delta L = 0$, and $\Delta G \neq 0$, then $\Delta F \neq 0$. If $\Delta L = 0$ and $\Delta G = 0$, then $\boldsymbol{x} \xrightarrow{\neq} \boldsymbol{y}$ is a burst of substitutions that does not involve any pattern, then $\Delta D \neq 0$, thus $\Delta F \neq 0$.

Let $n_{1^k}(\boldsymbol{x})$ be the number of appearances of 1^k in $\boldsymbol{x}$. For example, if $\boldsymbol{x} = 111100111$, then $n_{1^2}(\boldsymbol{x}) = 5$. We will show that n_{1^k} is our desired g. The next example provides the intuition. Note that $N_{1^k}(\boldsymbol{x})$ is the sum of $n_{1^k}(\boldsymbol{x}_i)$ over $\boldsymbol{x}^{\mathcal{P}} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n_{\mathcal{P}}(\boldsymbol{x})})$.

Example 5. *Let $N^L_{1^k}(\boldsymbol{x})$ and $N^R_{1^k}(\boldsymbol{x})$ denote the contribution of the left side and the right side with respect to the '|' symbols below. Similar definitions apply to $\boldsymbol{y}$. Below, we show two typical cases in which $\boldsymbol{x} \xrightarrow{\neq} \boldsymbol{y}$ is a burst of substitutions that shifts a pattern. In both cases, we show that $N_{1^k}(\boldsymbol{x}) \neq N_{1^k}(\boldsymbol{y})$, implying that $\Delta N_{1^k} \neq 0$.*

First, suppose $\boldsymbol{x} = \boldsymbol{\alpha} 0^\epsilon | 0^k 1^k \boldsymbol{\beta}$ and $\boldsymbol{y} = \boldsymbol{\alpha} | 0^k 1^k 1^\epsilon \boldsymbol{\beta}$, for some binary string $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. Here $0 < \epsilon \leq k$. The burst of substitutions occurs at positions $[|\boldsymbol{\alpha}| + k + 1, |\boldsymbol{\alpha}| + k + \epsilon]$. Below is an example with $k = 10, \epsilon = 7$.

$$\boldsymbol{x} = \boldsymbol{\alpha} \overbrace{0000000}^{0^\epsilon} | \overbrace{000\mathbf{0000000}}^{0^k} \overbrace{1111111111}^{1^k} \boldsymbol{\beta}$$
$$\boldsymbol{y} = \boldsymbol{\alpha} | \underbrace{0000000000}_{0^k} \underbrace{\mathbf{1111111111}}_{1^k} \underbrace{1111111}_{1^\epsilon} \boldsymbol{\beta}$$

In this case, we can see $N^L_{1^k}(\boldsymbol{x}) = N^L_{1^k}(\boldsymbol{y})$, and because of the $1^{k+\epsilon}$ in $\boldsymbol{y}$ compared to the 1^k in $\boldsymbol{x}$, we have $N^R_{1^k}(\boldsymbol{x}) = N^R_{1^k}(\boldsymbol{y}) - \epsilon$. So $N_{1^k}(\boldsymbol{x}) \neq N_{1^k}(\boldsymbol{y})$, and thus $\Delta N_{1^k} \neq 0$.

Second, suppose $\boldsymbol{x} = \boldsymbol{\alpha} | 0^k 1^k 0^{k-\epsilon} 1^k \boldsymbol{\beta}$ and $\boldsymbol{y} = \boldsymbol{\alpha} 0^k 1^{k-\epsilon} | 0^k 1^k \boldsymbol{\beta}$. Here $0 < \epsilon < k$ (when $\epsilon = k$, this edit is also described by the previous case). The burst of substitutions occurs at positions $[|\boldsymbol{\alpha}| + 2k - \epsilon + 1, |\boldsymbol{\alpha}| + 2k]$. An example with $k = 10, \epsilon = 7$ is shown below.

$$\boldsymbol{x} = \boldsymbol{\alpha} | \overbrace{0000000000}^{0^k} \overbrace{111\mathbf{1111111}}^{1^k} \overbrace{000}^{0^{k-\epsilon}} \overbrace{1111111111}^{1^k} \boldsymbol{\beta}$$
$$\boldsymbol{y} = \boldsymbol{\alpha} \underbrace{0000000000111}_{0^k 1^{k-\epsilon}} | \underbrace{\mathbf{0000000}000}_{0^k} \underbrace{1111111111}_{1^k} \boldsymbol{\beta}$$

In this case, we can see $N^L_{1^k}(\boldsymbol{x}) = N^L_{1^k}(\boldsymbol{y})$, and because there are two 1^k's between '|' and β in $\boldsymbol{x}$ compared to only one 1^k between '|' and β in $\boldsymbol{y}$, we have $N^R_{1^k}(\boldsymbol{x}) = N^R_{1^k}(\boldsymbol{y}) + 1$. So $N_{1^k}(\boldsymbol{x}) \neq N_{1^k}(\boldsymbol{y})$, and thus $\Delta N_{1^k} \neq 0$.

It can be shown that the two cases in the above example are the only two key cases in which a burst of substitutions shifts a pattern (see full version). We can now formally construct f. For a string $\boldsymbol{x}$, let both $l(\boldsymbol{x})$ and $L(\boldsymbol{x})$ represent the length of $\boldsymbol{x}$, let $n_{1^k}(\boldsymbol{x})$ be the number of 1^k inside $\boldsymbol{x}$, and let $d(\boldsymbol{x})$ be as in the second attempt. Let $B = 3(2^k)$, and $C = 40k(2^k)$, and define

$$f(\boldsymbol{x}) := d(\boldsymbol{x}) + Bn_{1^k}(\boldsymbol{x}) + Cl(\boldsymbol{x}). \tag{3}$$

If $\boldsymbol{x} \xrightarrow{\neq} \boldsymbol{y}$, then $\boldsymbol{f}(\boldsymbol{x}) \Rightarrow \boldsymbol{f}(\boldsymbol{y})$ satisfies the second condition of locatability. It is not hard to prove that $\boldsymbol{f}(\boldsymbol{x}) \Rightarrow \boldsymbol{f}(\boldsymbol{y})$ also satisfies the first condition of locatability (see full version). In summary, we have the following Lemma.

Lemma 6. *If $\boldsymbol{x} \xrightarrow{\neq} \boldsymbol{y}$, then $\boldsymbol{f}(\boldsymbol{y})$ is obtained from $\boldsymbol{f}(\boldsymbol{x})$ by a locatable 3-substring edit.*

Lemma 7. *If $\boldsymbol{x}$ is $(\mathcal{P}, \delta)$-dense and $\boldsymbol{x} \to \boldsymbol{y}$, then $\boldsymbol{f}(\boldsymbol{x})$ and $\boldsymbol{f}(\boldsymbol{y})$ are $8\delta C$-bounded strings.*

c) Correcting an edit with known approximate location: We start by adapting a result on systematic codes correcting multiple edits [12] to a hash correcting a k-substring edit, since a k-substring edit can also be viewed as k edits. The adapted result states that a substring edit in a string of length n can be corrected using $4k \log n + o(\log n)$ bits of redundancy. Although our goal is to achieve about $\log n$ redundancy, this result will be useful in our construction.

Lemma 8 (c.f. [12])**.** *There exists a hash $\phi_n : \{0,1\}^n \to \{0,1\}^{4k \log n + o(\log n)}$ that can correct a single k-substring edit. Namely, if $\boldsymbol{r} \in \{0,1\}^n$ and $\boldsymbol{r} \to \boldsymbol{s}$, then $\boldsymbol{r}$ can be recovered from $\boldsymbol{s}$ and $\phi_n(\boldsymbol{r})$.*

Let $\psi_A(\boldsymbol{x}) : \{0,1\}^n \to [2^{4k \log A + o(\log A)}]$ be defined as

$$\psi_A(\boldsymbol{x}) = \left(\left(\sum_{i \text{ odd}} \phi_A(\boldsymbol{x}_i) \right) \bmod 2^{4k \log A + o(\log A)}, \left(\sum_{i \text{ even}} \phi_A(\boldsymbol{x}_i) \right) \bmod 2^{4k \log A + o(\log A)} \right),$$

where the $\boldsymbol{x}_i$ result from partitioning $\boldsymbol{x}$ into parts of length A (note that in the definition of the the map $\psi_A(\boldsymbol{x})$ we are not partitioning $\boldsymbol{x}$ according to $\boldsymbol{x}^{\mathcal{P}}$).

Lemma 9 is an adaptation of a technique used in [7] to our substring edit scenario, which will allow us to correct a substring edit whose position is approximately known.

Lemma 9 (c.f. [7])**.** *If $\boldsymbol{x} \to \boldsymbol{y}$, then one can recover $\boldsymbol{x}$ given $\boldsymbol{y}$ and $\psi_A(\boldsymbol{x})$ and an interval of length A that contains the edit position.*

B. Substring edit-correcting code and its redundancy

We now put the components previously discussed together to construct the error-correcting code. Let $\mathcal{P}$ be as in Section III-A, and let f and C be the one in equation (3). Let

$$h(\boldsymbol{x}) := \Big(\mathrm{VT}(\boldsymbol{f}(\boldsymbol{x})) \bmod 2000Cn, \left(\sum_{i=1}^{n_{\mathcal{P}}(\boldsymbol{x})} f_i(\boldsymbol{x}) \right) \bmod 10Ck, n_{\mathcal{P}}(\boldsymbol{x}) \bmod 5 \Big).$$

For $c_1, c_2 \in \mathbb{N}$, let

$$\mathcal{C} = \{\boldsymbol{x} \text{ is } (\mathcal{P}, \delta)\text{-dense: } h(\boldsymbol{x}) = c_1, \psi_{O(\delta^2)}(\boldsymbol{x}) = c_2\}.$$

The next lemma will be used to show that the redundancy arising from restricting $\boldsymbol{x}$ to be $(\mathcal{P}, \delta)$-dense is negligible.

Lemma 10. *Let $\delta = k2^{2k+3} \log n$. For a uniformly randomly chosen $\boldsymbol{x} \in \{0,1\}^n$,*

$$\Pr[\boldsymbol{x} \textit{ is not } (\mathcal{P}, \delta)\textit{-dense}] < 1/n.$$

Theorem 11. *$\mathcal{C}$ corrects one k-substring edit. Moreover, for $\delta = k2^{2k+3} \log n$, it has redundancy $\log n + O(\log \log n)$.*

Proof sketch: We first prove the redundancy. Note that $h(\boldsymbol{x})$ needs $\log n + O(1)$ redundancy and $\psi_{O(\delta^2)}(\boldsymbol{x})$ needs $O(\log \log n)$ redundancy. By Lemma 10, restricting $\boldsymbol{x}$ to be $(\mathcal{P}, \delta)$-dense adds $O(1)$ redundancy. So $\mathcal{C}$ has redundancy $\log n + O(\log \log n)$.

To prove the error-correction ability of the code, we first show that if $\boldsymbol{x} \to \boldsymbol{y}$, given $\boldsymbol{y}$ and $h(\boldsymbol{x})$, we can determine whether $\boldsymbol{x} = \boldsymbol{y}$. By Lemma 6 and the definition of locatable substring edit, $\boldsymbol{x} = \boldsymbol{y}$ if and only if $\sum_{i=1}^{n_{\mathcal{P}}(\boldsymbol{x})} f_i(\boldsymbol{x}) \equiv \sum_{i=1}^{n_{\mathcal{P}}(\boldsymbol{y})} f_i(\boldsymbol{y}) \mod 10Ck$ and $n_{\mathcal{P}}(\boldsymbol{x}) \equiv n_{\mathcal{P}}(\boldsymbol{y}) \mod 5$. These two equations can be checked given $\boldsymbol{y}$ and $h(\boldsymbol{x})$ to determine whether $\boldsymbol{x} = \boldsymbol{y}$. If $\boldsymbol{x} = \boldsymbol{y}$, then we just recover $\boldsymbol{x}$ by outputting $\boldsymbol{y}$.

We will now assume $\boldsymbol{x} \xrightarrow{\neq} \boldsymbol{y}$ and show that we can determine the edit position to within an interval of length $O(\delta^2)$. By Lemmas 6 and 7, we can treat $\boldsymbol{f}(\boldsymbol{x})$ and $\boldsymbol{f}(\boldsymbol{y})$ as the $\boldsymbol{z}$ and $\boldsymbol{w}$ in Lemma 2, and determine the position of the locatable edit to within an interval of length $O(\delta)$. Since $\boldsymbol{x}$ is $(\mathcal{P}, \delta)$-dense, we can further determine the position of the original edit to within an interval of length $O(\delta^2)$. In the final step, by Lemma 9, we can recover $\boldsymbol{x}$ by $\psi_{O(\delta^2)}(\boldsymbol{x})$. ■

REFERENCES

[1] S. M. H. Tabatabaei Yazdi, Y. Yuan, J. Ma, H. Zhao, and O. Milenkovic, "A Rewritable, Random-Access DNA-Based Storage System," *Scientific Reports*, vol. 5, Sep. 2015. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4585656/

[2] 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.

[3] Y. Tang, S. Motamen, H. Lou, K. Whritenour, S. Wang, R. Gabrys, and F. Farnoud, "Correcting a substring edit error of bounded length," in *2023 IEEE International Symposium on Information Theory (ISIT)*, 2023, pp. 2720–2725.

[4] P. A. H. Bours, "Codes for correcting insertion and deletion errors," 1994.

[5] V. Levenshtein, "Asymptotically optimum binary code with correction for losses of one or two adjacent bits," *Problemy Kibernetiki*, vol. 19, pp. 293–298, 1967.

[6] 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.

[7] 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.

[8] R. Gabrys, V. Guruswami, J. Ribeiro, and K. Wu, "Beyond single-deletion correcting codes: Substitutions and transpositions," *IEEE Transactions on Information Theory*, vol. 69, no. 1, pp. 169–186, 2023.

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

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

[11] 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.

[12] 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.