Optimal k-Deletion Correcting Codes

Jin Sima and **Jehoshua Bruck**
Department of Electrical Engineering, California Institute of Technology, Pasadena 91125, CA, USA

***Abstract*—Levenshtein introduced the problem of constructing k-deletion correcting codes in 1966, proved that the optimal redundancy of those codes is $O(k \log N)$, and proposed an optimal redundancy single-deletion correcting code (using the so-called VT construction). However, the problem of constructing optimal redundancy k-deletion correcting codes remained open. Our key contribution is a solution to this longstanding open problem. We present a k-deletion correcting code that has redundancy $8k \log n + o(\log n)$ and encoding/decoding algorithms of complexity $O(n^{2k+1})$ for constant k.**

I. Introduction

A set of binary vectors of length N is a k-deletion code (denoted by $\mathcal{C}$) iff any two vectors in $\mathcal{C}$ do not share a subsequence of length $N-k$. The problem of constructing a k-deletion code was introduced by Levenshtein [1]. He proved that the optimal redundancy (defined as $N - \log|\mathcal{C}|$) is $O(k \log N)$. Specifically, it is in the range $k \log N + o(\log N)$ to $2k \log N + o(\log N)$. In addition, he proposed the following optimal construction (the well-known Varshamov-Tenengolts (VT) code [2]):

$$\left\{(c_1, \ldots, c_N) : \sum_{i=1}^{N} i c_i \equiv 0 \bmod (N+1)\right\}, \qquad (1)$$

that is capable of correcting a single deletion with redundancy not more than $\log(N+1)$ [1]. The encoding/decoding complexity of VT codes is linear in N. Generalizing the VT construction to correct more than a single deletion was elusive for more than 50 years. In particular, the past approaches [3] [4], [5] result in asymptotic code rates that are bounded away from 1.

A recent breakthrough paper [6] proposed a k-deletion code construction with $O(k^2 \log k \log N)$ redundancy and $O_k(N \log^4 N)$[1] encoding/decoding complexity. For the case $k = 2$ deletions, the redundancy was improved in [7], [8]. Specifically, the code in [8] has redundancy of $7 \log N$ and linear encoding/decoding complexity. The work in [9] considered correction with high probability and proposed a k-deletion code construction with redundancy $(k+1)(2k+1)\log N + o(\log N)$ and decoding complexity $O(N^{k+1}/\log^{k-1} N)$. This randomized coding setting was improved in [10], where redundancy $O(k \log(n/k))$ and complexity $poly(n,k)$ is achieved. However, finding a deterministic k-deletion code construction that achieves the optimal order redundancy $O(k \log N)$ remained elusive.

Our key contribution is a solution to this longstanding open problem: We present a code construction that achieves $O(k \log N)$ redundancy and $O(N^{2k+1})$ encoding/decoding computational complexity (note that the complexity is polynomial in N). The following theorem summarizes our main result. We note that throughout this paper, the optimality of a code is redundancy-wise rather than cardinality-wise. The problem of finding optimal cardinality k deletion code appears highly nontrivial even for $k = 1$.

Theorem 1. *For any integer $n > k$ and $N = n + 8k \log n + o(\log n)$, there exists an encoding function $\mathcal{E} : \{0,1\}^n \to \{0,1\}^N$, computed in $O(n^{2k+1})$ time, and a decoding function $\mathcal{D} : \{0,1\}^{N-k} \to \{0,1\}^n$, computed in $O(n^{k+1})$ time, such that for any $\mathbf{c} \in \{0,1\}^n$ and subsequence $\mathbf{d} \in \{0,1\}^{N-k}$ of $\mathcal{E}(\mathbf{c})$, we have that $\mathcal{D}(\mathbf{d}) = \mathbf{c}$.*

Recently, an independent work [11] proposed a k deletion code with $O(k \log n)$ redundancy and better complexity of $poly(n,k)$. Compare to the constant $8k \log n$ in our paper, the constant in [11] is not explicitly given and is at least $200k \log n$. Moreover, the approaches in [11] and this paper are different.

Next we identify and describe our key ideas. The key building blocks in our code construction are: (i) *generalizing the VT construction* to k deletions by considering constrained sequences, (ii) separating the encoded vector to blocks and using *concatenated codes* and (iii) a novel strategy to *separate the vector to blocks by a single pattern.*

In our previous work for 2-deletions codes [8], we *generalized the VT construction*. In particular, we proved that while the higher order parity checks $\sum_{i=1}^{n} i^j c_i \bmod (n^j+1)$, $j = 0, 1, \ldots, t$ might not work in general, those parity checks work in the two deletions case when the sequences are constrained to have no adjacent 1's. In this paper we generalize this idea, specifically, the higher order parity checks work for $k = t/2$ deletions when the sequences we need to protect satisfy the *following constraint:* The distance between any two adjacent 1's is at least k.

The fact that we can correct k deletions using the generalization of the VT construction on constrained sequences, enables a *concatenated code construction*, which separates the sequence $\mathbf{c}$ into small blocks. Each block is protected by an inner code, usually a k-deletion code. All the blocks together are protected by an outer code, for example, a Reed-Solomon code. Separating and identifying the boundaries between blocks is one of the main challenges in the concatenated code construction. The work in [12], [13] resolved this issue

The work was supported in part by NSF grants CCF-1816965 and CCF-1717884 .

[1]The notion O_k denotes *parameterized complexity*, i.e., $O_k(N \log^4 N) = f(k)O(N \log^4 N)$ for some function f.

by inserting markers between blocks. In [6], an approach that uses occurrences of short subsequences, called patterns, as markers was proposed. The success of decoding in existing approaches requires that the patterns can not be destroyed or generated by k deletions / insertions.

Here, we improve the redundancy in [6] by using *a single pattern to separate the blocks* and allowing it to be destroyed or generated by deletions / insertions. The pattern, which we call *synchronization pattern*, is a length $3k + \lceil \log k \rceil + 4$ sequence $\mathbf{a} = (a_1, \ldots, a_{3k+\lceil \log k \rceil + 4})$ satisfying

- $a_{3k+i} = 1$ for $i \in \{0, \ldots, \lceil \log k \rceil + 4\}$.
- There does not exist a $j \in \{1, \ldots, 3k-1\}$, such that $a_{j+i} = 1$ for $i \in \{0, \ldots, \lceil \log k \rceil + 4\}$.

Namely, a *synchronization pattern* is a sequence that ends with $\lceil \log k \rceil + 5$ consecutive 1's and no other 1 run with length $\lceil \log k \rceil + 5$ exists. For a sequence $\mathbf{c} = (c_1, \ldots, c_n)$, define a *synchronization vector* $\mathbb{1}_{sync}(\mathbf{c}) \in \{0,1\}^n$ by

$$\mathbb{1}_{sync}(\mathbf{c})_i = \begin{cases} 1, & \text{if } (c_{i-3k+1}, c_{i-3k+2}, \ldots, c_{i+\lceil \log k \rceil + 4}) \\ & \text{is a } \textit{synchronization pattern}, \\ 0, & \text{else.} \end{cases}$$

Note that $\mathbb{1}_{sync}(\mathbf{c})_i = 0$ for $i \in [1, 3k-1]$ and for $i \in [n - \lceil \log k \rceil - 3, n]$. It can be seen from the definition that any two consecutive 1 entries in $\mathbb{1}_{sync}(\mathbf{c})$ have distance at least $3k$.

Now we are ready to describe our construction that is a generalization of the VT code. Define the integer vectors

$$\mathbf{m}^{(\ell)} \triangleq (1^\ell, 1^\ell + 2^\ell, \ldots, \sum_{j=1}^{n} j^\ell)$$

for $\ell \in \{0, \ldots, 6k\}$, where the i-th entry of $\mathbf{m}^{(\ell)}$ is the sum of the ℓ-th powers of the first i entries. Given a sequence $\mathbf{c} \in \{0,1\}^n$ we compute a (VT like) redundancy of dimension $6k+1$ as follows:

$$f(\mathbf{c})_\ell \triangleq \mathbf{c} \cdot \mathbf{m}^{(\ell)} \bmod 3kn^{\ell+1}, \tag{2}$$

for $\ell \in \{0, \ldots, 6k\}$. It will be shown that the vector $f(\mathbb{1}_{sync}(\mathbf{c}))$ helps recover the *synchronization vector* $\mathbb{1}_{sync}(\mathbf{c})$ from k deletions in $\mathbf{c}$.

The rest of the paper is organized as follows. Section II provides an outline of our construction and some of the basic lemmas. Section III presents our VT generalization for recovering the *synchronization vector*. Section IV explains how to correct k deletions based on the *synchronization vector*, when the *synchronization patterns* appear frequently. Section V describes an algorithm to transform a sequence into one with dense *synchronization patterns*. Section VI presents the encoding and decoding of the code. Section VII concludes the paper.

II. Outline and Preliminaries

In this section we give an overview of the ingredients (Lemmas 1, 2, and 3) that constitute our code construction, as well as existing results (Lemmas 4, 5) that are needed in our proof. We first present a lemma showing how to recover *synchronization vector* from k deletions. The result, which will be proved in Section III, is crucial in our concatenated code construction. For a sequence $\mathbf{c} \in \{0,1\}^n$, define its deletion ball $B_k(\mathbf{c})$ to be the collection of sequences that share a length $n-k$ subsequence with $\mathbf{c}$. For a number vector $\mathbf{v} = (v_0, \ldots, v_{6k})$ that satisfies $0 \le v_\ell < 3kn^{\ell+1}$, let

$$M(\mathbf{v}) = \sum_{\ell=0}^{6k} v_\ell \prod_{i=0}^{\ell-1} 3kn^{i+1} \tag{3}$$

be a one-to-one mapping that maps the vector $\mathbf{v}$ into a number that ranges in $[0, (3k)^{6k+1} n^{(3k+1)(6k+1)} - 1]$, where the set $[a, b] = \{a, a+1, \ldots, b\}$, called an interval, consists of consecutive integers between a and b for $a \le b$.

Lemma 1. *For integers n and k, there exists a function $p : \{0,1\}^n \to [1, 2^{2k \log n + o(\log n)}]$, such that if $M(f(\mathbb{1}_{sync}(\mathbf{c}))) \equiv M(f(\mathbb{1}_{sync}(\mathbf{c}'))) \bmod p(\mathbf{c})$ for two sequence $\mathbf{c} \in \{0,1\}^n$ and $\mathbf{c}' \in B_k(\mathbf{c})$, then $\mathbf{c} = \mathbf{c}'$.*

With the knowledge of its *synchronization vector* $\mathbb{1}_{sync}(\mathbf{c})$, we show in the next lemma how to recover the sequence $\mathbf{c}$ with redundancy $O(k \log n)$, when the 0 runs in $\mathbb{1}_{sync}(\mathbf{c})$ is not long. We introduce a notion, called *k dense*, to characterize the limited 0 run length property.

A sequence $\mathbf{c} \in \{0,1\}^n$ is said to be *k dense* if the distance between any two consecutive 1 entries in $\mathbb{1}_{sync}(\mathbf{c})$ is at most

$$\begin{aligned} L \triangleq & (\lceil \log k \rceil + 5) 2^{\lceil \log k \rceil + 8} \lceil \log n \rceil \\ & + (3k + \lceil \log k \rceil + 4)(\lceil \log n \rceil + 9 + \lceil \log k \rceil) \end{aligned}$$

More precisely, the 0 runs in $\mathbb{1}_{sync}(\mathbf{c})$ have length at most $L-1$. The following lemma will be proved in Section IV.

Lemma 2. *For integers k and $n > k$, there exists a function $Hash_k : \{0,1\}^n \to \{0,1\}^{4k \log n + o(\log n)}$, such that every k* dense *sequence $\mathbf{c} \in \{0,1\}^n$ can be recovered from its length $n-k$ subsequence $\mathbf{d}$ and $Hash_k(\mathbf{c})$.*

Lemma 1 and Lemma 2 show how to protect *k dense* sequences. As the final building block, the following lemma presents a mapping that transforms any sequence to a *k dense* sequence. Its proof will be given in Section V. Based on Lemmas 1, 2, and 3, our k deletion code is given in Section VI.

Lemma 3. *For integers k and $n > k$, there exists a map $T : \{0,1\}^n \to \{0,1\}^{n+2\lceil \log k \rceil + 10}$, such that $T(\mathbf{c})$ is a k* dense *sequence for $\mathbf{c} \in \{0,1\}^n$. Moreover, the sequence $\mathbf{c}$ can be recovered from $T(\mathbf{c})$.*

Lemma 4 gives a k deletion correcting hash function that is computable in $O_k(poly(n))$ time. It is a slight variation of the result in [6]. Lemma 5 (See [14]) gives an upper bound on the number of divisors of a positive integer n. It will be used in proving Lemma 1.

Lemma 4. *Let k be a fixed integer. For integers M and n. There exists a hash function $H : \{0,1\}^M \to \{0,1\}^{\lceil (M/\lceil \log n \rceil) \rceil (2k \log\log n + O(1))}$, computable in $O_k((M/\log n) n \log^{2k} n)$ time, such that any sequence $\mathbf{c} \in \{0,1\}^M$ can be recovered from its length $M-k$ subsequence $\mathbf{d}$ and the hash $H(\mathbf{c})$.*

Lemma 5. *For a positive integer $n \geq 3$, the number of divisors of n is upper bounded by $2^{1.6 \ln n/(\ln \ln n)}$.*

III. Protecting the synchronization vectors

For a sequence $\mathbf{c} \in \{0,1\}^n$, let $g(\mathbf{c})$ be a dimension $6k+1$ vector with entries defined by

$$g(\mathbf{c})_\ell \triangleq \mathbf{c} \cdot \mathbf{m}^{(\ell)},$$

for $\ell \in \{0, \ldots, 6k\}$. The proof of Lemma 1 is based on the following two lemmas together with Lemma 5.

Lemma 6. *For $\mathbf{c}, \mathbf{c}' \in \{0,1\}^n$, if $\mathbf{c}' \in B_k(\mathbf{c})$, then $\mathbb{1}_{sync}(\mathbf{c}) \in B_{3k}(\mathbb{1}_{sync}(\mathbf{c}'))$.*

Let $\mathcal{R}_m$ be the set of length n sequences the 0 runs in which have length at least $m-1$, meaning that any two consecutive 1 entries in a sequence $\mathbf{c} \in \mathcal{R}_m$ have distance at least m.

Lemma 7. *For $\mathbf{c}, \mathbf{c}' \in \mathcal{R}_{3k}$, if $\mathbf{c}' \in B_{3k}(\mathbf{c})$, and $g(\mathbf{c}) = g(\mathbf{c}')$ then $\mathbf{c} = \mathbf{c}'$.*

Proof. We first compute the difference $g(\mathbf{c})_\ell - g(\mathbf{c}')_\ell$. Since $\mathbf{c}' \in B_{3k}(\mathbf{c})$, there exist two subsets $\boldsymbol{\delta} = \{\delta_1, \ldots, \delta_{3k}\} \subset \{1, \ldots, n\}$ and $\boldsymbol{\delta}' = \{\delta'_1, \ldots, \delta'_{3k}\} \subset \{1, \ldots, n\}$ such that deleting bits with positions $\boldsymbol{\delta}$ and $\boldsymbol{\delta}'$ respectively from $\mathbf{c}$ and $\mathbf{c}'$ result in the same length $n-3k$ subsequence. Further define $\boldsymbol{\Delta} = \{i : c_i = 1\}$ and $\boldsymbol{\Delta}' = \{i : c'_i = 1\}$ to be the positions of 1 entries in $\mathbf{c}$ and $\mathbf{c}'$ respectively. Let $S_1 = \boldsymbol{\Delta} \cap \boldsymbol{\delta}$ and $S_2 = \boldsymbol{\Delta} \cap ([1,n] \backslash \boldsymbol{\delta})$ be the sets of 1 entry positions that are deleted and not deleted in $\mathbf{c}$ respectively. Similarly let $S'_1 = \boldsymbol{\Delta}' \cap \boldsymbol{\delta}'$ and $S'_2 = \boldsymbol{\Delta}' \cap ([1,n] \backslash \boldsymbol{\delta}')$. Let the elements in $\boldsymbol{\delta} \cup \boldsymbol{\delta}'$ be ordered by $p_1 \leq p_2 \leq \ldots \leq p_{6k}$. Denoting $p_0 = 0$ and $p_{6k+1} = n$, we have that

$$\begin{aligned} g(\mathbf{c})_\ell - g(\mathbf{c}')_\ell &= \sum_{i \in \boldsymbol{\Delta}} \mathbf{m}_i^{(\ell)} - \sum_{i \in \boldsymbol{\Delta}'} \mathbf{m}_i^{(\ell)} \\ &= \sum_{j=0}^{6k} \sum_{i=p_j+1}^{p_{j+1}} (|S_1 \cap [p_{j+1}, n]| - |S'_1 \cap [p_{j+1}, n]| \\ &\quad + |S_2 \cap [i,n]| - |S'_2 \cap [i,n]|) i^\ell. \end{aligned} \tag{4}$$

In the following we show (a): $-1 \leq |S_2 \cap [i,n]| - |S'_2 \cap [i,n]| \leq 1$ for $i \in [1,n]$. and (b): For each interval $(p_j, p_{j+1}]$, $j = 0, \ldots, 6k$, we have either $|S_2 \cap [i,n]| - |S'_2 \cap [i,n]| \leq 0$ for all $i \in (p_j, p_{j+1}]$ or $|S_2 \cap [i,n]| - |S'_2 \cap [i,n]| \geq 0$ for all $i \in (p_j, p_{j+1}]$.

Note that each $i \in S_2$ corresponds uniquely to an index $i' \in S'_2$ so that c_i and $c'_{i'}$ end in the same position after deleting the bits with positions $\boldsymbol{\delta}$ and $\boldsymbol{\delta}'$ respectively from $\mathbf{c}$ and $\mathbf{c}'$, i.e., $i - |\boldsymbol{\delta} \cap [1, i-1]| = i' - |\boldsymbol{\delta}' \cap [1, i'-1]|$. Hence $|i' - i| \leq 3k$. Let $a = \min_{x \in S_2, x \in [i,n]} x$ and $b = \min_{x \in S'_2, x \in [i,n]} x$. Then any element in $S_2 \backslash \{a\}$ and $S'_2 \backslash \{b\}$ is at least $i + 3k$. It follows that every $x \in (S_2 \backslash \{a\})$ corresponds to some $y \in S'_2$. Similarly, every $y \in (S'_2 \backslash \{b\})$ corresponds to some $x \in S_2$. Therefore, we have that $-1 \leq |S_2 \cap [i,n]| - |S'_2 \cap [i,n]| \leq 1$ and (a) is proved.

We now prove (b) by contradiction. Supposed on the contrary, there exist $i_1, i_2 \in (p_j, p_{j+1}]$ such that $i_1 < i_2$ and $(|S_2 \cap [i_1,n]| - |S'_2 \cap [i_1,n]|)(|S_2 \cap [i_2,n]| - |S'_2 \cap [i_2,n]|)$. By symmetry it can be assumed that $|S_2 \cap [i_1,n]| - |S'_2 \cap [i_1,n]| = -1$ and $|S_2 \cap [i_2,n]| - |S'_2 \cap [i_2,n]| = 1$. This implies that $i_1 - |\boldsymbol{\delta} \cap [1, i_1 - 1]| > i_1 - |\boldsymbol{\delta}' \cap [1, i_1 - 1]|$ and $i_2 - |\boldsymbol{\delta} \cap [1, i_2 - 1]| < i_2 - |\boldsymbol{\delta}' \cap [1, i_2 - 1]|$, which are equivalent to $|\boldsymbol{\delta} \cap [1, i_1 - 1]| < |\boldsymbol{\delta}' \cap [1, i_1 - 1]|$ and $|\boldsymbol{\delta} \cap [1, i_2 - 1]| > |\boldsymbol{\delta}' \cap [1, i_2 - 1]|$ respectively. However, since $i_1, i_2 \in (p_j, p_{j+1}]$, we have that $|\boldsymbol{\delta} \cap [1, i_1 - 1]| = |\boldsymbol{\delta} \cap [1, i_2 - 1]|$ and $|\boldsymbol{\delta}' \cap [1, i_1 - 1]| = |\boldsymbol{\delta}' \cap [1, i_2 - 1]|$, which results in a contradiction. Hence (b) is proved. Denote

$$\begin{aligned} s_i \triangleq &|S_1 \cap [i,n]| - |S'_1 \cap [i,n]| \\ &+ |S_2 \cap [i,n]| - |S'_2 \cap [i,n]| \end{aligned}$$

From (a) and (b) it follows that for each interval $(p_j, p_{j+1}]$, $j \in \{0, \ldots, 6k\}$, either $s_i \geq 0$ for all $i \in (p_j, p_{j+1}]$ or $s_i \leq 0$ for all $i \in (p_j, p_{j+1}]$. Let $\mathbf{x} = (x_0, \ldots, x_{6k}) \in \{-1, 1\}^{6k+1}$ be a vector defined by

$$x_i = \begin{cases} -1, & \text{if } s_j < 0 \text{ for some } j \in (p_i, p_{i+1}] \\ 1, & \text{else.} \end{cases}.$$

Then from Eq. (4) we have that

$$g(\mathbf{c})_\ell - g(\mathbf{c})_\ell = \sum_{j=0}^{6k} \Big(\sum_{i=p_j+1}^{p_{j+1}} |s_i| i^\ell \Big) x_j \tag{5}$$

Let A be a $6k+1 \times 6k+1$ matrix with its entries defined by $A_{e,j} = \sum_{i=p_{j-1}+1}^{p_j} |s_i| i^{e-1}$ for $e, j \in \{1, \ldots, 6k+1\}$. If $g(\mathbf{c}) = g(\mathbf{c}')$, we have the following linear equation

$$A\boldsymbol{x} = \begin{bmatrix} \sum_{i=p_0+1}^{p_1} |s_i| i^0 & \cdots & \sum_{i=p_{6k}+1}^{p_{6k+1}} |s_i| i^0 \\ \vdots & \ddots & \vdots \\ \sum_{i=p_0+1}^{p_1} |s_i| i^{6k} & \cdots & \sum_{i=p_{6k}+1}^{p_{6k+1}} |s_i| i^{6k} \end{bmatrix} \begin{bmatrix} x_0 \\ \vdots \\ x_{6k} \end{bmatrix} = 0 \tag{6}$$

We show that this is impossible unless A is a zero matrix. Suppose on the contrary that A is nonzero, let $j_1 < \ldots < j_Q$ be the indices of all nonzero columns of A. Let B be a submatrix of A obtained by choosing the first Q rows and columns with indices $j_1, \ldots, j_Q$. Then we have that

$$B\boldsymbol{x}' = \begin{bmatrix} \sum_{i=p_{j_1-1}+1}^{p_{j_1}} |s_i| i^0 & \cdots & \sum_{i=p_{j_Q-1}+1}^{p_{j_Q}} |s_i| i^0 \\ \vdots & \ddots & \vdots \\ \sum_{i=p_{j_1-1}+1}^{p_{j_1}} |s_i| i^Q & \cdots & \sum_{i=p_{j_Q-1}+1}^{p_{j_Q}} |s_i| i^Q \end{bmatrix} \begin{bmatrix} x_{j_1} \\ \vdots \\ x_{j_Q} \end{bmatrix} = 0$$

Denote the interval $\mathcal{I}_i = (p_{j_{i-1}}, p_{j_i}]$. By the multi-linearity of the determinant and by the determinant formula of the Vandermonde matrix,

$$\begin{aligned} \det(B) &= \sum_{i_1 \in \mathcal{I}_1, \ldots, i_Q \in \mathcal{I}_Q} \prod_{q=1}^{Q} |s_{i_q}| \begin{bmatrix} i_1^0 & \cdots & i_Q^0 \\ \vdots & \ddots & \vdots \\ i_1^Q & \cdots & i_Q^Q \end{bmatrix} \\ &= \sum_{i_1 \in \mathcal{I}_1, \ldots, i_Q \in \mathcal{I}_Q} \prod_{q=1}^{Q} |s_{i_q}| \prod_{1 \leq m_1 < m_2 \leq Q} (i_{m_2} - i_{m_1}) \end{aligned}$$

is positive since $i_{m_2} > i_{m_1}$ for $m_2 > m_1$ and there exist $i_1 \in \mathcal{I}_1, \ldots, i_Q \in \mathcal{I}_Q$ such that $|s_{i_1}|, \ldots, |s_{i_Q}| > 0$. Therefore,

the linear equation $B\boldsymbol{x}' = 0$ does not have nonzero solutions, contradicting to the fact that $\boldsymbol{x}' \in \{-1, 1\}^Q$. Hence A is a zero matrix, meaning that

$$\begin{aligned}&|S_1 \cap [i,n]| - |S_1' \cap [i,n]| + |S_2 \cap [i,n]| - |S_2' \cap [i,n]| \\ =&|\boldsymbol{\Delta} \cap [i,n]| - |\boldsymbol{\Delta}' \cap [i,n]| = 0\end{aligned}$$

for $i \in \{1, \ldots, n\}$. This implies $\boldsymbol{\Delta} = \boldsymbol{\Delta}'$ and thus $\mathbf{c} = \mathbf{c}'$. □

Let $\boldsymbol{\Delta} = \{i : \mathbb{1}_{sync}(\mathbf{c})_i = 1\}$ and $\boldsymbol{\Delta}' = \{i : \mathbb{1}_{sync}(\mathbf{c}')_i = 1\}$. From Lemma 6 we have that $\mathbb{1}_{sync}(\mathbf{c}') \in B_{3k}(\mathbb{1}_{sync}(\mathbf{c}))$. Hence $(\mathbb{1}_{sync}(\mathbf{c}')_i, \ldots, \mathbb{1}_{sync}(\mathbf{c}')_n) \in B_{3k}((\mathbb{1}_{sync}(\mathbf{c})_i, \ldots, \mathbb{1}_{sync}(\mathbf{c})_n))$. This implies that $||\boldsymbol{\Delta} \cap [i,n]| - |\boldsymbol{\Delta}' \cap [i,n]|| \leq 3k$. Therefore,

$$\begin{aligned}&|g(\mathbb{1}_{sync}(\mathbf{c}))_\ell - g(\mathbb{1}_{sync}(\mathbf{c}'))_\ell| \\ =&|\sum_i^n (|\boldsymbol{\Delta} \cap [i,n]| - |\boldsymbol{\Delta}' \cap [i,n]|) i^\ell| \leq \sum_i^n 3ki^\ell < 3kn^{\ell+1}.\end{aligned} \tag{7}$$

Hence if $f(\mathbb{1}_{sync}(\mathbf{c})) = f(\mathbb{1}_{sync}(\mathbf{c}'))$ (see (2) for definition of f), we have that $g(\mathbb{1}_{sync}(\mathbf{c}))_\ell \equiv g(\mathbb{1}_{sync}(\mathbf{c}'))_\ell \bmod 3kn^{\ell+1}$, which implies that $g(\mathbb{1}_{sync}(\mathbf{c})) = g(\mathbb{1}_{sync}(\mathbf{c}'))$ according to E.q. (7). Since $\mathbb{1}_{sync}(\mathbf{c}') \in B_{3k}(\mathbb{1}_{sync}(\mathbf{c}))$ and $\mathbb{1}_{sync}(\mathbf{c}), \mathbb{1}_{sync}(\mathbf{c}') \in \mathcal{R}_{3k}$, from Lemma 7 we have that $\mathbb{1}_{sync}(\mathbf{c}) = \mathbb{1}_{sync}(\mathbf{c}')$.

We are now ready to prove Lemma 1. Since $f(\mathbb{1}_{sync}(\mathbf{c})) \neq f(\mathbb{1}_{sync}(\mathbf{c}'))$ for $\mathbf{c}' \in B_k(\mathbf{c}) \backslash \{\mathbf{c}\}$, we have that $|M(f(\mathbb{1}_{sync}(\mathbf{c}))) - M(f(\mathbb{1}_{sync}(\mathbf{c}')))| \neq 0$ (see (3) for definition of M) for $\mathbf{c}' \in B_k(\mathbf{c}) \backslash \{\mathbf{c}\}$. According to Lemma 5, the number of divisors of $|M(f(\mathbb{1}_{sync}(\mathbf{c}))) - M(f(\mathbb{1}_{sync}(\mathbf{c})))|$ is upper bounded by $2^{2[(3k+1)(6k+1)\ln n + (6k+1)\ln 3k]/\ln((3k+1)(6k+1)\ln n + (6k+1)\ln 3k)} = 2^{o(\log n)}$. Since $|B_k(\mathbf{c})| \leq \binom{n}{k}^2 2^k \leq 2n^{2k}$, there are at most $2n^{2k}2^{o(\log n)}$ numbers that divide $|M(f(\mathbb{1}_{sync}(\mathbf{c}))) - M(f(\mathbb{1}_{sync}(\mathbf{c}')))|$ for some $\mathbf{c}' \in B_k(\mathbf{c}) \backslash \{\mathbf{c}\}$. Therefore, there exists a number $p(\mathbf{c}) \in [1, 2^{2k\log n + o(\log n)}]$ such that $p(\mathbf{c}) \nmid |M(f(\mathbb{1}_{sync}(\mathbf{c}'))) - M(f(\mathbb{1}_{sync}(\mathbf{c})))|$ for $\mathbf{c}' \in B_k(\mathbf{c}) \backslash \{\mathbf{c}\}$. Hence, if $M(f(\mathbb{1}_{sync}(\mathbf{c}'))) \equiv M(f(\mathbb{1}_{sync}(\mathbf{c}))) \bmod p(\mathbf{c})$ and $\mathbf{c}' \in B_k(\mathbf{c})$, we have that $M(f(\mathbb{1}_{sync}(\mathbf{c}'))) - M(f(\mathbb{1}_{sync}(\mathbf{c}))) \equiv 0 \bmod p(\mathbf{c})$ and thus $\mathbf{c}' = \mathbf{c}$.

IV. Hash for k *dense* sequences

In this section, we present a hash function for correcting k deletions in a k *dense* sequence $\mathbf{c}$, based on the knowledge of the *synchronization vector* $\mathbb{1}_{sync}(\mathbf{c})$.

Let the positions of the 1 entries in $\mathbb{1}_{sync}(\mathbf{c})$ be ordered by $t_1 < t_2 < \ldots < t_J$, where $J = \sum_{i=1}^n \mathbb{1}_{sync}(\mathbf{c})_i$. Furthermore, let $t_0 = 0$ and $t_{J+1} = n + 1$ Split $\mathbf{c}$ into blocks $\mathbf{a}_0, \ldots, \mathbf{a}_J$, where

$$\mathbf{a}_j = (c_{t_j+1}, c_{t_j+2}, \ldots, c_{t_{j+1}-1}).$$

Let the hash function $Hash_k$: be given by

$$Hash_k(\mathbf{c}) = RS_{2k}((H(\mathbf{a}_0), \ldots, H(\mathbf{a}_J))),$$

where $RS_{2k}(\mathbf{c})$ is the redundancy of a systematic Reed-Solomon code that corrects $2k$ substitution errors. The sequence $(H(\mathbf{a}_0), \ldots, H(\mathbf{a}_J))$ is a sequence of symbols $H(\mathbf{a}_j)$ (see Lemma 4), each having alphabet size $2^{\lceil (L/\lceil \log n \rceil) \rceil (2k \log\log n + O(1))}$. The length of $Hash_k(\mathbf{c})$ is $\max\{4k\log n, 4k\lceil (L/\lceil \log n \rceil) \rceil (2k\log\log n + O(1))\} = 4k\log n + o(\log n)$. We now present the following procedure that recovers $\mathbf{c}$ from its length $n-k$ subsequence $\mathbf{d}$ and the hash function $Hash_k(\mathbf{c})$, given $\mathbb{1}_{sync}(\mathbf{c})$.

1) **Step 1:** Find the *synchronization vector* $\mathbb{1}_{sync}(\mathbf{d}) \in \{0,1\}^{n-k}$ of $\mathbf{d}$. Find the locations of 1 entries in $\mathbb{1}_{sync}(\mathbf{c})$ and order them by $t_1 < \ldots < t_J$. Let $t_0 = 0$ and $t_{J+1} = n + 1$
2) **Step 2:** Let $\mathbb{1}_{sync}(\mathbf{d})_0 = \mathbb{1}_{sync}(\mathbf{d})_{n+1} = 1$. For each $j \in [0, J]$, if there exist two numbers $i_j \in [t_j - k, t_j]$ and $i_{j+1} \in [t_{j+1} - k, t_{j+1}]$ such that $\mathbb{1}_{sync}(\mathbf{d})_{i_j} = \mathbb{1}_{sync}(\mathbf{d})_{i_j} = 1$, set $\mathbf{a}_j' = (d_{i_j+1}, d_{i_j+2}, \ldots, d_{i_{j+1}-1})$. Else set $\mathbf{a}_j' = 0$.
3) **Step 3:** Apply the Reed-Solomon decoder to decode $(H(\mathbf{a}_0'), \ldots, H(\mathbf{a}_J'), Hash_k(\mathbf{c}))$ and to recover $H(\mathbf{a}_j)$ for $j \in [0, J]$.
4) **Step 4:** Let $\mathbf{b}_j = (d_{t_j+1}, \ldots, d_{t_{j+1}-k-1})$, recover $\mathbf{a}_j$ by using $\mathbf{b}_j$ and $H(\mathbf{a}_j)$.

To prove the correctness of the decoding, we first show that $\mathbf{a}_j$ can be recovered from $\mathbf{b}_j$ and $H(\mathbf{a}_j)$. This can be done by noticing that $(d_{t_j+1}, \ldots, d_{t_{j+1}-k-1})$ is a length $|\mathbf{a}_j| - k$ subsequence of $\mathbf{a}_j$, where $|\mathbf{a}_j|$ is the length of $\mathbf{a}_j$. Furthermore, it can be proved that there exist at most $2k$ indices j, such that $\mathbf{a}_j' \neq \mathbf{a}_j$. Thus the Reed-Solomon code works.

V. Transformation to k *dense* sequences

In this section we present an algorithm to compute $T(\mathbf{c})$, which transforms any sequence $\mathbf{c} \in \{0,1\}^n$ into a k *dense* sequence. Let $\mathbf{1}^x$ and $\mathbf{0}^y$ denote consecutive x 1's and consecutive y 0's respectively. It can be shown that any sequence $\mathbf{c}$ satisfying the following is a k *dense* sequence.

Property 1: There is no $i \in [1, n]$ that satisfies $(c_j, c_{j+1}, \ldots, c_{j+\lceil \log k \rceil + 4}) \neq \mathbf{1}^{\lceil \log k \rceil + 5}$ for $j \in [i, i + L_1 - \lceil \log k \rceil - 5]$, where $L_1 \triangleq (\lceil \log k \rceil + 5)2^{\lceil \log k \rceil + 8} \lceil \log n \rceil$.

property 2: Any interval $[i, i + L_2 - 1] \subseteq [1, n]$ of length $L_2 \triangleq (3k + \lceil \log k \rceil + 4)(\lceil \log n \rceil + 9 + \lceil \log k \rceil)$ contains a sub-interval $[j, j + 3k + \lceil \log k \rceil + 3]$, such that $(c_m, c_{m+1}, \ldots, c_{m+\lceil \log k \rceil + 4}) \neq \mathbf{1}^{\lceil \log k \rceil + 5}$ for $m \in [j, j + 3k - 1]$.

Next we show how to transform a sequence into one that satisfies Properties 1 and 2. The following two lemmas will be used, where Lemma 8 presents a function that outputs a sequence satisfying Property 1.

Lemma 8. *For integers k and $n > k$, there exists a map $T_1 : \{0,1\}^n \to \{0,1\}^{n + \lceil \log k \rceil + 5}$, computable in $O(n^2 k \log n \log k)$ time, such that $T_1(\mathbf{c})$ satisfies Property 1. The sequence $\mathbf{c}$ can be recovered from $T_1(\mathbf{c})$.*

Lemma 9. *For an integer k, let $\mathbf{c} \in \{0,1\}^{3k + \lceil \log k \rceil + 4}$ be a sequence such that $c_i = c_{i+1} = \ldots = c_{i + \lceil \log k \rceil + 4} = 1$ for some $i \in [1, 3k]$. There exists a mapping $T_2 : \{0,1\}^{3k + \lceil \log k \rceil + 4} \to \{0,1\}^{3k + \lceil \log k \rceil + 3}$, computable in $O(k^2 \log k)$ time, such that $T_2(\mathbf{c})$ contains no $\lceil \log k \rceil + 5$ consecutive 1 bits. In addition, the sequence $\mathbf{c}$ can be recovered from $T_2(\mathbf{c})$.*

We are now ready to give the encoding and decoding procedure for computing $T(\mathbf{c})$. The encoding procedure for computing $T(\mathbf{c})$ is as follows

1) **Initialization:** Let $T(\mathbf{c}) = T_1(\mathbf{c})$. Append $\mathbf{1}^{\lceil \log k \rceil + 5}$ to the end of the sequence $T(\mathbf{c})$. Let $n' = n + \lceil \log k \rceil + 5$ and $i = 1$. Go to Step 1.
2) **Step 1:** If $i \leq n' - \lceil \log k \rceil - 5$ and for every $j \in [i, i + L_2 - 3k - \lceil \log k \rceil - 4]$, there exists $m \in [j, j + 3k - 1]$ such that $(c_m, c_{m+1}, \ldots, c_{m+\lceil \log k \rceil + 4}) = \mathbf{1}^{\lceil \log k \rceil + 5}$. Split $(c_i, c_{i+1}, \ldots, c_{i+L_2-1})$ into $(\lceil \log n \rceil + 9 + \lceil \log k \rceil)$ blocks $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_{\lceil \log n \rceil + 9 + \lceil \log k \rceil}$ of length $3k + \lceil \log k \rceil + 4$. Delete $(\mathbf{b}_2, \ldots, \mathbf{b}_{\lceil \log n \rceil + 8 + \lceil \log k \rceil})$ from $T(\mathbf{c})$ and append $(0, T_2(\mathbf{b}_2), T_2(\mathbf{b}_3), \ldots, T_2(\mathbf{b}_{\lceil \log n \rceil + 8 + \lceil \log k \rceil}), i + 3k + \lceil \log k \rceil + 4, \mathbf{1}^{\lceil \log k \rceil + 5}, 0)$ to the end of $T(\mathbf{c})$. Let $n' = n' - L_2 + 6k + 2\lceil \log k \rceil + 8$ and $i = 1$. Repeat. Else go to Step 2.
3) **Step 2:** If $i \leq n'$, let $i = i + 1$ and go to Step 1. Else output $T(\mathbf{c})$.

The length of $T(\mathbf{c})$ remains to be $n + 2\lceil \log k \rceil + 10$.

Finally, we show that $T(\mathbf{c})$ is decodable, with the following decoding procedure that recovers $\mathbf{c}$ from $T(\mathbf{c})$.

1) **Initialization:** Let $\mathbf{c} = T(\mathbf{c})$ and go to Step 1.
2) **Step 1:** If $c_{n+2\lceil \log k \rceil + 10} = 0$, let i be the decimal representation of $(c_{n+\lceil \log k \rceil + 5 - \lceil \log n \rceil}, \ldots, c_{n+\lceil \log k \rceil + 4})$. Break $(c_{n+4\lceil \log k \rceil + 19 - L_2 + 6k}, \ldots, c_{n+\lceil \log k \rceil + 4 - \lceil \log n \rceil})$ into $(L_2 - 6k - 2\lceil \log k \rceil - 8)/(3k + \lceil \log k \rceil + 4)$ blocks $\mathbf{b}'_1, \ldots, \mathbf{b}'_{(L_2 - 6k - 2\lceil \log k \rceil - 8)/(3k + \lceil \log k \rceil + 4)}$ of length $3k + \lceil \log k \rceil + 3$. Compute $\mathbf{b}_j = T_2^{-1}(\mathbf{b}'_j)$ for $j \in [1, (L_2 - 6k - 2\lceil \log k \rceil - 8)/(3k + \lceil \log k \rceil + 4)]$, where $T_2^{-1}(\mathbf{b}'_j)$ is obtained by applying T_2 decoder (Lemma 9) on $\mathbf{b}'_j$. Delete $(c_{n+2\lceil \log k \rceil + 11 - L_2 + 6k}, \ldots, c_{n+2\lceil \log k \rceil + 10})$ from $\mathbf{c}$ and insert $\mathbf{b}_1, \ldots, \mathbf{b}_{(L_2 - 6k - 2\lceil \log k \rceil - 8)/(3k + \lceil \log k \rceil + 4)}$ at location i of $\mathbf{c}$. Repeat. Else output $\mathbf{c}$.

VI. Encoding

In this section we present the encoding function $\mathcal{E}$ and prove Theorem 1. The function $\mathcal{E}$ is given by

$$\mathcal{E}(\mathbf{c}) = (T(\mathbf{c}), R'(\mathbf{c}), R''(\mathbf{c})),$$

where

$$\begin{aligned} R'(\mathbf{c}) =& (M(f(\mathbb{1}_{sync}(T(\mathbf{c})))) \bmod p(T(\mathbf{c})), p(T(\mathbf{c})), \\ & Hash_k(T(\mathbf{c}))), \text{ and} \\ R''(\mathbf{c}) =& Rep_{k+1}(H(R'(\mathbf{c}))). \end{aligned}$$

Here M is the function defined in Eq. (3) and $Rep_{k+1}(H(R'(\mathbf{c})))$ is the $k+1$-fold repetition of the bits in $H(R'(\mathbf{c}))$. It can be seen that the codeword $\mathcal{E}(\mathbf{c})$ has length $N = n + 8k \log n + o(\log n)$. Thus the redundancy is $8k \log n + o(\log n)$.It can then be shown that

(a). The redundancy $R'(\mathbf{c})$ can be recovered from k deletions with the help of $R''(\mathbf{c})$.

(b). The sequence $\mathbf{c}$ can be recovered from k deletions with the help of $R'(\mathbf{c})$.

Let N_1 and N_2 be the length of $R'(\mathbf{c})$ and $R''(\mathbf{c})$ repectively. To decode $\mathbf{c}$ from a $\mathbf{d}$, it suffices to note that (1). The sequence $(d_1, \ldots, d_{n+2\lceil \log k \rceil + 10 - k})$ is a length $n + 2\lceil \log k \rceil + 10 - k$ subsequence $T(\mathbf{c}) \in \{0,1\}^{2\lceil \log k \rceil + 10}$. (2). The sequence $(d_{n+2\lceil \log k \rceil + 11}, d_{n+2\lceil \log k \rceil + 10 + N_1 - k})$ is a length $N_1 - k$ subsequence of $R'(\mathbf{c})$. (3). The sequence $(d_{n+2\lceil \log k \rceil + 11 + N_1}, \ldots, d_{n+2\lceil \log k \rceil + 10 + N_1 + N_2 - k})$ is a length $N_2 - k$ subsequence of $R''(\mathbf{c})$. Since $R''(\mathbf{c})$ is a $k+1$-fold repetition of $H(R'(\mathbf{c}))$, it can be recovered from its length $N_2 - k$ subsequence.

The encoding complexity of $\mathcal{E}(\mathbf{c})$ is $O(n^{2k+1})$ for using brute force to find $p(T(\mathbf{c}))$. The decoding complexity is $O(n^{k+1})$ for using brute force to recover $\mathbb{1}_{sync}(T(\mathbf{c}))$ from $M(f(\mathbb{1}_{sync}(T(\mathbf{c})))) \bmod p(T(\mathbf{c}))$ and $p(T(\mathbf{c}))$.

VII. Conclusion and Future Work

We construct a k-deletion correcting code with optimal order redundancy. Interesting open problems include finding complexity $O(N^{O(1)})$ encoding/decoding algorithms for our proposed code, as well as constructing a systematic k-deletion code with optimal redundancy.

References

[1] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet physics doklady*, vol. 10, no. 8, pp. 707–710, 1966.

[2] R. R. Varshamov and G. M. Tenengolts, "Codes which correct single asymmetric errors," *Autom. Remote Control*, vol. 26, no. 2, pp. 286–290, 1965.

[3] A. S. Helberg and H. C. Ferreira, "On multiple insertion/deletion correcting codes," *IEEE Trans. on Inf. Th.*, vol. 48, no. 1, pp. 305–308, 2002.

[4] K. A. Abdel-Ghaffar, F. Paluncic, H. C. Ferreira, and W. A. Clarke, "On Helberg's generalization of the Levenshtein code for multiple deletion/insertion error correction," *IEEE Trans. on Inf. Th.*, vol. 58, no. 3, pp. 1804–1808, 2012.

[5] F. Paluncic, K. A. Abdel-Ghaffar, H. C. Ferreira, and W. A. Clarke, "A multiple insertion/deletion correcting code for run-length limited sequences," *IEEE Trans. on Inf. Th.*, vol. 58, no. 3, pp. 1809–1824, 2012.

[6] J. Brakensiek, V. Guruswami, and S. Zbarsky, "Efficient low-redundancy codes for correcting multiple deletions," *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms* (SODA), pp. 1884–1892, 2016

[7] R. Gabrys and F. Sala, "Codes correcting two deletions." *IEEE Int. Symp. on Inform. Theory.*, Vail, USA, pp. 426–430, 2018.

[8] J. Sima, N. Raviv, and J. Bruck, "Two Deletion Correcting Codes from Indicator Vectors," *IEEE Int. Symp. on Inform. Theory.*, Vail, USA, pp. 421–425, 2018.

[9] S. K. Hanna and S. El Rouayheb, "Guess & check codes for deletions, insertions, and synchronization," *IEEE Trans. on Inf. Th.*, vol. 65, no. 1, pp. 3–15, 2019.

[10] B. Haeupler, "Optimal document exchange and new codes for small number of insertions and deletions." *arXiv:1804.03604* [cs.DS], 2018.

[11] K. Cheng, Z. Jin, X. Li and K. Wu, "Deterministic document exchange protocols, and almost optimal binary codes for edit errors," *IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 200–211, 2018.

[12] L. J. Schulman and D. Zuckerman, "Asymptotically good codes correcting insertions, deletions, and transpositions," *IEEE Trans. on Inf. Th.*, vol. 45, no. 7, pp. 2552–2557, 1999.

[13] V. Guruswami and C. Wang, "Deletion codes in the high-noise and high-rate regimes," *IEEE Trans. on Inf. Th.*, vol. 63, no. 4, pp. 1961–1970, 2017.

[14] J. L. Nicolas, "On highly composite numbers," *Ramanujan revisited, Proceedings of the centenary conference, University of Illinois at Urbana-Champaign,* pp. 215-244, 1987.