Robust Indexing – Optimal Codes for DNA Storage

Jin Sima[1], **Netanel Raviv**[2] and **Jehoshua Bruck**[1]

[1]Department of Electrical Engineering, California Institute of Technology, Pasadena 91125, CA, USA
[2]Department of Computer Science and Engineering, Washington University in Saint Louis, St. Louis 63130, MO, USA

***Abstract*—The channel model of encoding data as a set of unordered strings is receiving great attention as it captures the basic features of DNA storage systems. However, the challenge of constructing optimal redundancy codes for this channel remained elusive. In this paper, we solve this open problem and present an order-wise optimal construction of codes that correct multiple substitution errors for this channel model. The key ingredient in the code construction is a technique we call robust indexing: it assigns indices to unordered strings (hence, creating order) and embeds information in the indices (eliminating unnecessary redundancy). In addition, our robust indexing technique can be applied to the construction of optimal deletion/insertion codes for this channel.**

I. Introduction

The interest in storing data in synthetic DNA is drastically increasing lately, due to its advantages of ultra high data density and longevity over other storage media. Tremendous progress has been made in synthesizing and sequencing technologies, which brings about a new era in large-scale DNA storage. Prototype implementations of DNA storage stored 643KB data in [3] and 739KB data in [6] respectively, which were later improved in [4] and [20]. A recent work [12] achieved 200MB storage of data.

In DNA storage systems, data is represented by strings of four nucleotides that make up the synthesized DNA molecules. One of the key features that distinguishes DNA storage from conventional storage media is that data are encoded as an unordered set of short strings, rather than a single long string. This is since current technology cannot synthesize a single DNA string long enough to encode the entire data. The typical length of a short DNA string is several hundreds.

When writing the data, these short strings are synthesized into DNA molecules and after a Polymerase Chain Reaction (PCR) process, which amplifies the number of copies of each DNA molecule, are stored in a DNA pool. When reading the data, the DNA pool is sampled and sequenced, producing multiple reads of the short strings that encode the data. In the reading and writing process, sequencing errors and synthesizing errors might occur, resulting in substitution, deletion and insertion errors in the DNA strings. One way to correct these errors is to cluster the erroneous reads and use a sequence reconstruction algorithm to recover the strings. Yet such algorithms at the decoder cannot correct writing (synthesis) errors, which are amplified by the PCR process and cause the reconstructed strings to be erroneous. Thus, we need to construct error correcting codes for DNA storage, which is the focus of this paper.

While many coding theoretic works have been done for various channel models concerning different physical aspects of DNA storage [8], [5], [13], and [7], this paper focuses on coding over unordered sets, which captures some basic features in the writing and reading processes described above. Specifically, consider encoding data into M strings of length L. The decoder wishes to recover the data from erroneous versions of the M strings, which contain substitution, deletion and insertion errors.

This model has been extensively investigated recently. The work of [10] proposed constructions and upper bounds in terms of number of erroneous strings and the maximum number of errors in each string. To deal with unordered strings, one of the natural approaches is to assign $\log M$ bits to each string for indexing such that the strings are ordered. Such index-based construction was considered in [11] and [15], which attempted to correct errors in the indices. It was proved in [16] from an information theoretic view that index-based coding achieves the capacity of an unordered set of binary symmetric channel.

From a coding theoretic view, index-based codes require $O(M)$ bits of redundancy [10] and is suboptimal for small number of errors. The work of [17] showed that for a constant number K of substitution errors, the optimal redundancy has the order $O(K \log ML)$ and an explicit code with $O(K^2 \log ML)$ bits of redundancy was given. The problem of designing codes correcting a constant number of substitutions was also studied in [19], from a generalized Hamming distance perspective. Yet no order-wise optimal code construction for substitution errors was given.

In this paper, we propose order-wise optimal code constructions that achieve $O(K \log ML)$ redundancy for K substitution errors, based on a technique called *robust indexing*. Our main result is as follows

Theorem 1. *For integers M, L, and K, let $L' \triangleq 3\log M + 4K^2 + 1$. If $L' + 4KL' + 2K\log(4KL') \leq L$, then there exists an explicit K-substitution code, computable in $poly(M, L, K)$ time, that has redundancy $2K \log ML + (12K+2)\log M + O(K^3) + O(K \log\log ML)$.*

Instead of assigning index directly as in index-based coding, we *embed information into the index*. Note that to combat errors, the index bits themselves must form a substitution code and information is carried through choices of the code. Our *robust indexing* algorithm generates indexing bits in a greedy manner and has polynomial complexity. Furthermore,

this algorithm also applies to deletion/insertion errors with slight modification. With the recent progress in K-deletion codes [18], [2], we obtain a code that corrects K deletions with $O(K \log ML)$ redundancy, which will be given in the full version of this paper..

Theorem 2. *For integers M, L, K, and $L' \triangleq 3\log M + 4K^2 + 1$, if $L' + 4KL' + 2K\log(4KL') \leq L$, then there exists a K-deletion code, computable in $poly(M, L)$ time, that has redundancy $8K \log ML + (12K+2)\log M + O(K^3) + o(\log ML)$.*

The rest of the paper is organized as follows. Section II presents the notations and channel model. In Section III we provide an order-wise optimal code construction for substitution errors, and the robust indexing algorithm is given in Section IV. Section V concludes this paper.

II. Preliminaries

We focus on the binary alphabet $\{0,1\}$. For a set S and an integer m, denote by $\binom{S}{m}$ the family of all sets of m different elements in S, and by $\binom{S}{\leq m} = \bigcup_{i=1}^{M} \binom{S}{m}$ the family of all subsets of S with at most m elements. For an integer ℓ, let $\{0,1\}^{\leq \ell}$ be the set of all binary strings of length at most ℓ. In our channel model, it is assumed that the data is given as a binary string and encoded as an unordered set of M different strings $\{\mathbf{x}_i\}_{i=1}^{M}$ of length L. Hence, in this paper, a codeword refers to a set $\{\mathbf{x}_i\}_{i=1}^{M} \in \binom{\{0,1\}^L}{M}$, rather than a vector as in classic coding theoretic settings. Each element $\mathbf{x}_i$ in a codeword is referred to as a string. The assumption that the strings $\mathbf{x}_i$, $i \in [M]$, in a codeword are different stems from the fact that sequencing procedures cannot detect repeated strings in the codeword $\{\mathbf{x}_i\}_{i=1}^{M}$ by counting the frequency of each string in the sample. Moreover, as we can see from the definition of code redundancy that will be presented later, the asymptotic redundancy of a code is not affected by allowing repeated string in the codeword, whenever $M = o(2^L)$.

The codeword $\{\mathbf{x}_i\}_{i=1}^{M}$ is subject to substitution, deletion and insertion errors. In this paper, we propose codes for correcting substitution errors, and codes for deletion error will appear in future versions of this paper. A K-substitution error is an operation that flips at most K bits in the codeword. Each bit flip can occur in any of the strings $\mathbf{x}_i$, $i \in [M]$, where $[M] \triangleq \{1, \ldots, M\}$. A K-substitution error may cause two strings $\mathbf{x}_i$ and $\mathbf{x}_j$, $i, j \in [M]$ to be equal. As a result, the codeword $\{\mathbf{x}_i\}_{i=1}^{M}$ might turn into a set of less than M strings after a K-substitution-error. For any string set $\{\mathbf{x}_i\}_{i=1}^{M} \in \binom{\{0,1\}^L}{M}$, define its Hamming ball $\mathcal{B}_K^H(\{\mathbf{x}_i\}_{i=1}^{M}) \subseteq \binom{\{0,1\}^L}{\leq M}$ as the set of all possible all possible words (that is, sets) that result from a K substitution error in $\{\mathbf{x}_i\}_{i=1}^{M}$. A K substitution code $\mathcal{C}^H$ is an ensemble of codewords $\{\mathbf{x}_i\}_{i=1}^{M} \in \binom{\{0,1\}}{M}$ such that for any $S_1, S_2 \in \mathcal{C}^H$, we have that $\mathcal{B}_K^H(S_1) \cap \mathcal{B}_K^H(S_2) = \emptyset$. The redundancy of a K substitution code $\mathcal{C}^H$ is defined as $r(\mathcal{C}^H) = \binom{2^L}{M} - \log |\mathcal{C}^H|$.

Our code constructions make use of the well-known Reed-Solomon code, which is capable of correcting k substitutions in a length n codeword over an alphabet of size q, with $2k \log q$ bits redundancy, as long as $q \geq n-1$ [14]. Moreover, combinatorial numbering maps [9] are used in the robust indexing algorithm. Specifically, for integers m and n, there exist a map $F_{com} : [\binom{n}{m}] \to \binom{[n]}{m}$ that maps an integer $d \in [\binom{n}{m}]$ to a set of m different elements in $[n]$.

III. Robust indexing for codes over sets

In this section we describe our constructions of codes correcting substitution errors and prove Theorem 1. Our codes have redundancy $O(K \log ML)$, which is order-wise optimal whenever K is at most $O(\min\{L^{1/3}, L/\log M\})$.

Since the codewords consist of unordered strings, we assign indexing bits to each string such that order is induced lexicographically. However, instead of directly assigning the indices $1, \ldots, M$ to each string, we embed information into the indexing bits. In other words, we use the information bits themselves for the purpose of indexing. This provides greater efficiency in sending information.

Specifically, for a codeword $W = \{\mathbf{x}_i\}_{i=1}^{M}$, we choose the first L' bits $(x_{i,1}, x_{i,2}, \ldots, x_{i,L'})$, $i \in [M]$ in each string $\mathbf{x}_i$ as the indexing bits, and encode information in them. Then, the strings $\{\mathbf{x}_i\}_{i=1}^{M}$ are sorted according to the lexicographic order π of the indexing bits $(x_{i,1}, x_{i,2}, \ldots, x_{i,L'})$, $i \in [M]$, where $(x_{\pi(i),1}, x_{\pi(i),2}, \ldots, x_{\pi(i),L'}) < (x_{\pi(j),1}, x_{\pi(j),2}, \ldots, x_{\pi(j),L'})$ for $i < j$. Once $\{\mathbf{x}_i\}_{i=1}^{M}$ are ordered, it suffices to use a Reed-Solomon code to protect the concatenated string $(\mathbf{x}_{\pi(1)}, \ldots, \mathbf{x}_{\pi(M)})$, and thus the codeword $\{\mathbf{x}_i\}_{i=1}^{M}$, from K substitution errors.

One of the key issues with this approach is that the indexing bits and their lexicographic order can be disrupted by substitution errors. To deal with this, we present a technique referred to as robust indexing, which protects the indexing bits from substitution errors. The basic ideas of robust indexing are as follows: (1) Constructing the indexing bits $\{(x_{i,1}, x_{i,2}, \ldots, x_{i,L'})\}_{i=1}^{M}$ such that the Hamming distance between any two distinct $(x_{i,1}, x_{i,2}, \ldots, x_{i,L'})$ and $(x_{j,1}, x_{j,2}, \ldots, x_{j,L'})$ is at least $2K+1$, i.e., the strings $\{(x_{i,1}, x_{i,2}, \ldots, x_{i,L'})\}_{i=1}^{M}$ form a error correcting code under classic coding theoretic definition. Then, we can identify which string among $\{(x_{i,1}, x_{i,2}, \ldots, x_{i,L'})\}_{i=1}^{M}$ results in the erroneous version $(x'_{i,1}, x'_{i,2}, \ldots, x'_{i,L'})$, by using a minimum Hamming distance criterion; (2) Using additional redundancy to protect the set of indexing bits $\{(x_{i,1}, x_{i,2}, \ldots, x_{i,L'})\}_{i=1}^{M}$ from substitution errors. Note that we encode data in the code $\{(x_{i,1}, x_{i,2}, \ldots, x_{i,L'})\}_{i=1}^{M}$ through different choices of the code. After substitution errors, two choices of code, which represents different messages, might result in the same read $\{(x_{i,1}, x_{i,2}, \ldots, x_{i,L'})\}_{i=1}^{M}$.

Example 1. *For $K = M = 2$ and $L = 8$, consider two codes $\{11111111, 00000000\}$ and $\{11111111, 00010010\}$. Both have minimum Hamming distance greater than $2K+1 = 5$ and can result in the same set $\{11111111, 00000011\}$ after $K = 2$ substitutions.*

Hence, to recover the indexing bits $(x_{i,1}, x_{i,2}, \ldots, x_{i,L'})$, $i \in [M]$, we need to know

the code $\{(x_{i,1}, x_{i,2}, \ldots, x_{i,L'})\}_{i=1}^{M}$ to which the erroneous string $(x'_{i,1}, x'_{i,2}, \ldots, x'_{i,L'})$ is corrected, $i \in [M]$.

For an integer ℓ, let $\mathbb{1}_\ell$ be the all 1's vector of length ℓ. Define $\mathcal{S}^H$ as the set of all length L' codes with cardinality M and minimum Hamming distance at least $2K+1$, which contain $\mathbb{1}_{L'}$, that is,

$$\mathcal{S}^H \triangleq \{\{\mathbf{a}_1, \ldots, \mathbf{a}_M\} \in \binom{\{0,1\}^{L'}}{M} \Big| \mathbf{a}_1 = \mathbb{1}_{L'} \text{ and } d_H(\mathbf{a}_i, \mathbf{a}_j) \geq 2K+1 \text{ for every distinct } i, j \in [M]\}.$$

The following lemma provides a lower bound on the size of $\mathcal{S}^H$ and is obtained using counting arguments.

Lemma 1. *Let $Q = \sum_{i=0}^{2K} \binom{L'}{i}$ be the size of a Hamming ball of radius $2K$ centered at a vector in $\{0,1\}^{L'}$. We have that*

$$|\mathcal{S}^H| \geq \frac{(2^{L'} - MQ)^{M-1}}{(M-1)!}. \tag{1}$$

According to (1), there exists an invertible mapping $F_S^H : [\lceil \frac{(2^{L'}-MQ)^{M-1}}{(M-1)!} \rceil] \to \binom{\{0,1\}^{L'}}{M}$, computed in $O(2^{ML'})$ time using brute force, that maps an integer $d \in \left[\lceil \frac{(2^{L'}-MQ)^{M-1}}{(M-1)!} \rceil\right]$ to a code $F_S^H(d) \in \mathcal{S}^H$. A polynomial time algorithm that computes $F_S^H(d)$ will be given in the next section. Let us assume for now that the mapping F_S^H is given.

For a set $S \in \binom{\{0,1\}^{L'}}{\leq M}$, define the characteristic vector $\mathbb{1}(S) \in \{0,1\}^{2^{L'}}$ of S by

$$\mathbb{1}(S)_i = \begin{cases} 1 & \text{if the binary presentation of } i \text{ is in } S \\ 0 & \text{else} \end{cases}.$$

Notice that the Hamming weight of $\mathbb{1}(S)$ is M for every $S \in \binom{\{0,1\}^{L'}}{M}$. The following lemma is easily proved.

Lemma 2. *For $S_1, S_2 \in \binom{\{0,1\}^{L'}}{\leq M}$, if $S_1 \in \mathcal{B}_K^H(S_2)$, then $d_H(\mathbb{1}(S_1), \mathbb{1}(S_2)) \leq 2K$, where $d_H(\mathbb{1}(S_1), \mathbb{1}(S_2))$ is the Hamming distance between $\mathbb{1}(S_1)$ and $\mathbb{1}(S_2)$.*

We are ready to present the code construction. We use a set $S \in \mathcal{S}^H$ as indexing bits and protect the vector $\mathbb{1}_S$ from substitution errors. Note that any two strings in the set S have Hamming distance at least $2K+1$. Hence, knowing the set S, each string of indexing bits can be extracted from its erroneous version using a minimum distance decoder, which finds the unique string in S that is within Hamming distance K from it. The details are given as follows.

Consider the data $\mathbf{d} \in D$ to be encoded as a tuple $\mathbf{d} = (d_1, \mathbf{d}_2)$, where $d_1 \in [\lceil \frac{(2^{L'}-MQ)^{M-1}}{(M-1)!} \rceil]$ and

$$\mathbf{d}_2 \in \{0,1\}^{M(L-L')-4KL'-2K\lceil \log ML \rceil}.$$

Given $(d_1, \mathbf{d}_2)$, the codeword $\{\mathbf{x}_i\}_{i=1}^M$ is generated by the following procedure.

Encoding:

(1) Let $F_S^H(d_1) = \{\mathbf{a}_1, \ldots, \mathbf{a}_M\} \in \mathcal{S}^H$ such that $\mathbf{a}_1 = \mathbb{1}_{L'}$ and the $\mathbf{a}_i$'s are sorted in a descending lexicographic order. Let $(x_{i,1}, \ldots, x_{i,L'}) = \mathbf{a}_i$, for $i \in [M]$.

(2) Let

$$(x_{1,L'+1}, \ldots, x_{1,L'+4KL'}) = RS_{2K}(\mathbb{1}(\{\mathbf{a}_1, \ldots, \mathbf{a}_M\})),$$

where $RS_{2K}(\mathbb{1}(\{\mathbf{a}_1, \ldots, \mathbf{a}_M\}))$ is the redundancy of a systematic Reed-Solomon code that corrects $2K$ substitutions in $\mathbb{1}(\{\mathbf{a}_1, \ldots, \mathbf{a}_M\})$.

(3) Place the information bits of $\mathbf{d}_2$ in bits

$$(x_{1,L'+4KL'+1}, \ldots, x_{1,L}),$$
$$(x_{M,L'+1}, \ldots, x_{M,L-2K\lceil \log ML \rceil}); \text{ and}$$
$$(x_{i,L'+1}, \ldots, x_{i,L}) \text{ for } i \in [2, M-1].$$

(4) Define

$$\mathbf{m} = (\mathbf{x}_1, \ldots, \mathbf{x}_{M-1}, (x_{M,1}, \ldots, x_{M,L-2K\lceil \log ML \rceil}))$$

and let $(x_{M,L-2K\lceil \log ML \rceil+1}, \ldots, x_{M,L}) = RS_K(\mathbf{m})$, which is the Reed-Solomon redundancy that corrects K substitution errors in $\mathbf{m}$. Note that $(\mathbf{x}_1, \ldots, \mathbf{x}_M) = (\mathbf{m}, RS_K(\mathbf{m}))$ is a K-substitution correcting Reed-Solomon code.

(5) Output $\{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$.

Upon receiving the erroneous version[1] $\{\mathbf{x}'_1, \ldots, \mathbf{x}'_M\}$, the decoding procedure is as follows.

Decoding:

(1) Note that during the encoding process, the redundancy bits that correct the vector $\mathbb{1}(\{\mathbf{a}_i\}_{i=1}^M)$, i.e., the characteristic vector of the set of indexing bits $\{(x_{i,1}, \ldots, x_{i,L'})\}_{i=1}^M$, are stored in $\mathbf{x}_1$. Hence we must first identify the erroneous copy of $\mathbf{x}_1$. To this end, find the unique string $\mathbf{x}'_{i_0}$ such that $(x'_{i_0,1}, \ldots, x'_{i_0,L'})$ has at least $L'-K$ many 1-entries. Since the strings $\{\mathbf{x}_i\}_{i=1}^M$ have Hamming distance at least $2K+1$, there is a unique such string, which is the erroneous copy of $\{(x_{1,1}, \ldots, x_{1,L'})\}_{i=1}^M$. Hence $\mathbf{x}'_{i_0}$ is an erroneous copy of $\mathbf{x}_1$ and the string

$$(x'_{i_0,L'+1}, \ldots, x'_{i_0,L'+4KL})$$

is an erroneous copy of $(x_{1,L'+1}, \ldots, x_{1,L'+4KL'}) = RS_{2K}(\mathbb{1}(\{\mathbf{a}_i\}_{i=1}^M))$.

(2) According to Lemma 2, the vector $\mathbb{1}(\{(x_{i,1}, \ldots, x_{i,L'})\}_{i=1}^M)$ is within Hamming distance $2K$ from the vector $\mathbb{1}(\{(x'_{i,1}, \ldots, x'_{i,L'})\}_{i=1}^M)$. Hence the Hamming distance between

$$\mathbf{s}_1 = (\mathbb{1}(\{(x'_{i,1}, \ldots, x'_{i,L'})\}_{i=1}^M), (x'_{i_0,L'+1}, \ldots, x'_{i_0,L'+4KL})) \text{ and}$$
$$\mathbf{s}_2 = (\mathbb{1}(\{\mathbf{a}_i\}_{i=1}^M), RS_{2K}(\mathbb{1}(\{\mathbf{a}_i\}_{i=1}^M)))$$

is at most $2K$. Since $\mathbf{s}_2$ is a codeword in a Reed-Solomon code of minimum distance $2K$, it can be recovered from $\mathbf{s}_1$ using the Reed-Solomon decoder. Recover $d_1 = (F_S^H)^{-1}(\{\mathbf{a}_i\}_{i=1}^M)$.

(3) Since $\mathbf{s}_2$ is recovered, the strings $\{(x_{i,1}, \ldots, x_{i,L'})\}_{i=1}^M = \{\mathbf{a}_i\}_{i=1}^M$ are known. Sort $\{(x_{i,1}, \ldots, x_{i,L'})\}_{i=1}^M$

[1] Since the strings $\{\mathbf{x}_i\}_{i=1}^M$ have distance at least $2K+1$ with each other, the strings $\{\mathbf{x}'_i\}_{i=1}^M$ are different.

lexicographically in descending order. For each $i \in [M]$, find the unique $\pi(i) \in [M]$ such that $d_H((x'_{\pi(i),1}, \ldots, x'_{\pi(i),L'}), (x_{i,1}, \ldots, x_{i,L'})) \leq K$ (note that $i_0 = \pi(1)$). Similar to Step (1), we conclude that the string $\mathbf{x}'_{\pi(i)}$ is an erroneous copy of $\mathbf{x}_i$, $i \in [M]$, since the Hamming distance between $\mathbf{x}_j$ and $\mathbf{x}_i$ is at least $2K+1$ for $j \neq i$. Hence, the identities of $\{(x_{i,1}, \ldots, x_{i,L'})\}_{i=1}^M$ are determined from $\{(x'_{i,1}, \ldots, x'_{i,L'})\}_{i=1}^M$.

(4) Since $\mathbf{x}'_{\pi(i)}$ is an erroneous copy of $\mathbf{x}_i$, $i \in [M]$. it follows that the concatenation $\mathbf{s}' = (\mathbf{x}'_{\pi(1)}, \ldots, \mathbf{x}'_{\pi(M)})$ is an erroneous copy of $(\mathbf{x}_1, \ldots, \mathbf{x}_M) = (\mathbf{m}, RS_K(\mathbf{m}))$, where $\mathbf{m}$ is defined in Step (4) in the encoding procedure. Therefore, $(\mathbf{x}_1, \ldots, \mathbf{x}_M)$ and thus $\mathbf{d}_2$ can be recovered from $(\mathbf{x}'_{\pi(1)}, \ldots, \mathbf{x}'_{\pi(M)})$ by using the Reed-Solomon decoder.

(5) Output $(d_1, \mathbf{d}_2)$.

Therefore, the codeword $\{\mathbf{x}_i\}_{i=1}^M$ can be recovered. The redundancy of the code is

$$\begin{aligned} r(\mathcal{C}) &\leq \log \binom{2^L}{M} - \log\lceil \frac{(2^{L'} - MQ)^{M-1}}{(M-1)!} \rceil \\ &\quad - [M(L-L') - 4KL' - 2K\lceil \log ML \rceil] \\ &\leq 2K \log ML + (12K+2)\log M \\ &\quad + O(K^3) + O(K \log\log ML), \end{aligned} \tag{2}$$

The complexity of the encoding/decoding is dominated by that of decoding each individual index, which is $poly(M, L')$ by using brute force, and that of computing the function F_S^H, which as will be discussed in Section IV, is $poly(M, L, K)$.

IV. Computing F_S^H in polynomial time

In this section we present a polynomial time algorithm to compute the function F_S^H and thus complete the code construction in Section III. The result is as follows.

Theorem 3. *For integers M, L, K, $L' \triangleq 3\log M + 4K^2 + 1$ and $Q = \sum_{i=0}^{2K} \binom{L'}{i}$, there exists an invertible mapping $F_S^H : \left[\binom{2^{L'} - (M-1)Q + M - 1}{M-1}\right] \to \binom{\{0,1\}^{L'}}{M}$, computable in $poly(M, L)$ time, such that for any $d \in [\lceil \frac{(2^{L'} - MQ)^{M-1}}{(M-1)!} \rceil]$, we have that $F_S^H(d) \in \mathcal{S}^H$.*

The algorithm has a greedy flavor in the sense that the strings $\mathbf{a}_1, \ldots, \mathbf{a}_M$ are generated sequentially and each string $\mathbf{a}_i$, $i \in [2, M]$ is generated bit by bit. The algorithm consists of two steps. In the first step we map the integer $d \in [\lceil \frac{(2^{L'} - MQ)^{M-1}}{(M-1)!} \rceil]$ into $M-1$ integers $q_1, \ldots, q_M \in [2^{L'}]$ such that $q_1 = 2^{L'}$ and $q_{i+1} \leq q_i - Q$ for $i \in [M-1]$. In the second step, we use q_i to generate $\mathbf{a}_i$ sequentially for $i \in [2, M]$. The first step is given in the following lemma, which can be proved using the function F_{com}.

Lemma 3. *There exists an invertible map $F_Q^H : [\lceil \frac{(2^{L'} - MQ)^{M-1}}{(M-1)!} \rceil] \to [2^{L'}]^M$, computable in $poly(L', M)$ time, that maps and integer $d \in [\lceil \frac{(2^{L'} - MQ)^{M-1}}{(M-1)!} \rceil]$ to an integer tuple $(q_1, \ldots, q_M)$ such that $q_1 = 2^{L'}$ and $q_{i+1} \leq q_i - Q$ for $i \in [M-1]$.*

We now turn to the second step. Given the integers $F_Q^H(d) = (q_1, \ldots, q_M)$, we generate the indexing bits $\{\mathbf{a}_i = (x_{i,1}, \ldots, x_{i,L'})\}_{i=1}^M \in \mathcal{S}^H$. First, we have that $\mathbf{a}_1 = \mathbb{1}_{L'}$. The algorithm generates the indexing string $\mathbf{a}_i$ sequentially for $i \in [2, M]$. Each indexing string $\mathbf{a}_i$ is generated bit by bit in a recursive manner. We first give the following definition, on which the algorithm is based.

For a set of strings $A \subset \{0,1\}^{L'}$ and a string $\mathbf{a} \in \{0,1\}^\ell$ of length $\ell \in [L']$. Let

$$\begin{aligned} &N_H(\mathbf{a}, A) \\ &= \sum_{\mathbf{c}: \mathbf{c} \in A} |\{\mathbf{c}' : (c'_1, \ldots, c'_\ell) = \mathbf{a} \text{ and } d_H(\mathbf{c}', \mathbf{c}) \leq 2K\}| \end{aligned}$$

be the sum of the number of sequences that have prefix $\mathbf{a}$ and have Hamming distance at most $2K$ from $\mathbf{c}$, over $\mathbf{c} \in A$. The number $N_H(\mathbf{a}, A)$ has the following properties that will be useful in our proof. The first property implies that

$$\begin{aligned} 2^{L'-\ell} - N_H(\mathbf{a}, A) &= (2^{L'-\ell-1} - N_H((\mathbf{a}, 0), A)) \\ &\quad + (2^{L'-\ell-1} - N_H((\mathbf{a}, 1), A)), \end{aligned} \tag{3}$$

where $(\mathbf{a}, 0)$ or $(\mathbf{a}, 1)$ is the concatenation of $\mathbf{a}$ and a 0 or 1 bit respectively. Eq. (3) enables a recursion to generate each sequence $\mathbf{a}_i$. The second property provides a way to compute $N_H(\mathbf{a}, A)$.

Lemma 4. 1) *For any sequence $\mathbf{a} \in \{0,1\}^\ell$ of length $\ell \in [L'-1]$ and set $A \subset \{0,1\}^{L'}$, we have*

$$N_H(\mathbf{a}, A) = N_H((\mathbf{a}, 0), A) + N_H((\mathbf{a}, 1), A). \tag{4}$$

2) *For any $\mathbf{a} \in \{0,1\}^\ell$ and $A \subset \{0,1\}^{L'}$, we have*

$$N_H(\mathbf{a}, A) = \sum_{\mathbf{c}: \mathbf{c} \in A} \sum_{i=0}^{2K - d_H(\mathbf{a}, (c_1, \ldots, c_\ell))} \binom{L'-\ell}{i}. \tag{5}$$

Next, we present the algorithm that takes $F_Q^J(d) = (q_1, \ldots, q_M)$ as input and outputs $\mathbf{a}_i$ such that the decimal presentation $\text{decimal}(\mathbf{a}_i)$ of $\mathbf{a}_i, i \in [M]$ satisfies

$$\begin{aligned} \text{decimal}(\mathbf{a}_i) = \; & q_i - 1 + \\ & \sum_{\ell: a_{i,\ell} = 1 \text{ and } \ell \in [L']} N_H((a_{i,1}, \ldots, a_{i,\ell-1}, 0), \{\mathbf{a}_j\}_{j=1}^{i-1}). \end{aligned} \tag{6}$$

We then show that the sequences $\mathbf{a}_i, i \in [M]$ satisfying (6) are decodable, i.e., we can recover the tuple $(q_1, \ldots, q_M)$ from $\{\mathbf{a}_i\}_{i=1}^M$. Finally, we prove that $\{\mathbf{a}_i\}_{i=1}^M \in \mathcal{S}^H$.

Encoding:

for $i \in [M]$, do
$q = q_i$
  for $\ell \in [L']$, do
    if $2^{L'-\ell} - N_H((a_{i,1}, \ldots, a_{i,\ell-1}, 0), \{\mathbf{a}_j\}_{j=1}^{i-1}) \geq q$,
      then $a_{i,\ell} = 0$.
    else

$q = q - (2^{L'-\ell} -$
$N_H((a_{i,1}, \ldots, a_{i,\ell-1}, 0), \{\mathbf{a}_j\}_{j=1}^{i-1}))$
$a_{i,\ell} = 1$.
end if
end for
end for
return $\{\mathbf{a}_1, \ldots, \mathbf{a}_M\}$

The generation of $\mathbf{a}_i, i \in [M]$ in the encoding procedure can be intuitively characterized as walking on a complete binary tree of $L'+1$ layers. The walk starts at layer 1, i.e., the root of the binary tree, and ends at layer $L'+1$ at one of the leaf nodes. At each step, it goes to one of its two child nodes, which represent the bits 0 and 1 respectively. Each string $\mathbf{a}_i, i \in [M]$ is represented by the path of a walk. The paths are chosen so that the newly created $\mathbf{a}_i$ maintains Hamming distance of at least 2K from the former $\mathbf{a}_j$'s. For each path $\mathbf{a}_i = (a_{i,1}, \ldots, a_{i,L'})$ and each layer $\ell \in [L']$, assign the weight $w(a_{i,\ell}) = 2^{L'-\ell} - N_H((a_{i,1}, \ldots, a_{i,\ell}), \{\mathbf{a}_j\}_{j=1}^{i-1})$ to node $a_{i,\ell}$ in the ℓ-th layer, and the weight $w(\bar{a}_{i,\ell}) = 2^{L'-\ell} - N_H((a_{i,1}, \ldots, 1-a_{i,\ell}), \{\mathbf{a}_j\}_{j=1}^{i-1})$ to the sibling of node $a_{i,\ell}$. From Eq. (4) we have that $w(a_{i,\ell}) = w(a_{i,\ell+1}) + w(\bar{a}_{i,\ell+1})$ for $\ell \in [L'-1]$. Moreover, we have that $0 < q \leq w(a_{i,\ell})$ after the ℓ-th inner for loop in the i-th outer for loop. This is formalized in the following lemma, which can be used to prove that Eq. (6) holds and that $\{\mathbf{a}_1, \ldots, \mathbf{a}_M\} \in \mathcal{S}^H$.

Lemma 5. *After the ℓ-th inner for loop in the i-th outer for loop in the encoding procedure, $\ell \in [L']$, $i \in [M]$, we have*

$$0 < q \leq 2^{L'-\ell} - N_H((a_{i,1}, \ldots, a_{i,\ell}), \{\mathbf{a}_j\}_{j=1}^{i-1}) \quad (7)$$

At the end of the i-th outer for loop, we have that $q = 1$.

We now show that the strings $\{\mathbf{a}_1, \ldots, \mathbf{a}_M\}$ generated in the encoding procedure belong to $\mathcal{S}_H$. By Lemma 5, we have

$$q = 2^{L'-L'} - N_H(\mathbf{a}_i, \{\mathbf{a}_j\}_{j=1}^{i-1}) = 1,$$

at the end of each outer for loop in the encoding procedure. This implies that $N_H(\mathbf{a}_i, \{\mathbf{a}_j\}_{j=1}^{i-1}) = 0$ and thus $d_H(\mathbf{a}_i, \mathbf{a}_j)$ for $i \in [2, M]$ and $j \in [i-1]$. Moreover, since $q_1 = 2^{L'}$, we have that $\mathbf{a}_1 = \mathbb{1}_{L'}$. Therefore, $\{\mathbf{a}_i\}_{i=1}^M \in \mathcal{S}_H$.

Lemma 5 can be used to show the following lemma.

Lemma 6. *The output $\{\mathbf{a}_i\}_{i=1}^M$ of the encoding algorithm satisfies Eq. (6).*

Lemma 6 immediately implies a decoding algorithm that transforms $\{\mathbf{a}_i\}_{i=1}^M$ back to $(q_1, \ldots, q_M)$.

Decoding:

(1) Order the strings $\{\mathbf{a}_i\}_{i=1}^M$ such that $\mathbf{a}_1 > \mathbf{a}_2 > \ldots > \mathbf{a}_M$.
(2) For $i \in [M]$, compute q_i using Eq. (6).

To show that the decoding is correct, we prove that the strings $\mathbf{a}_1, \ldots, \mathbf{a}_M$ generated in the encoding procedure satisfy

$$\mathbf{a}_1 > \mathbf{a}_2 > \ldots > \mathbf{a}_M. \quad (8)$$

Then we conclude that the string $\mathbf{a}_i$ obtained by ordering $\{\mathbf{a}_i\}_{i=1}^M$ in Step (1) in the decoding procedure satisfies Eq. (6). Hence q_i, $i \in [M]$ can be recovered. Suppose on the contrary, there exist $\mathbf{a}_{i_1} > \mathbf{a}_{i_2}$ for some $i_1 > i_2$. Let ℓ^* be the most significant bit where $\mathbf{a}_{i_1}$ and $\mathbf{a}_{i_2}$ differ, i.e., $(a_{i_1,1}, \ldots, a_{i_1,\ell^*-1}) = (a_{i_2,1}, \ldots, a_{i_2,\ell^*-1})$ and $a_{i_1,\ell^*} = 1$ and $a_{i_2,\ell^*} = 0$. Then according to the if statement in the encoding procedure, we have that

$$q_{i_1} - \sum_{\ell: a_{i_1,\ell}=1 \text{ and } \ell \in [\ell^*]} (2^{L'-\ell} - N_H((a_{i_1,1}, \ldots, a_{i_1,\ell-1}, 0), \{\mathbf{a}_j\}_{j=1}^{i_1-1})) > 0 \text{ and}$$
$$q_{i_2} - \sum_{\ell: a_{i_1,\ell}=1 \text{ and } \ell \in [\ell*]} (2^{L'-\ell} - N_H((a_{i_1,1}, \ldots, a_{i_1,\ell-1}, 0), \{\mathbf{a}_j\}_{j=1}^{i_2-1})) \leq 0,$$

which implies that $q_{i_2} - q_{i_1} < (i_1 - i_2)Q$. This contradicts to the fact that the integers $(q_1, \ldots, q_M) = F_Q^H(d)$ satisfy $q_i - q_{i+1} > Q$ for $i \in [M-1]$, which implies $q_{i_1} - q_{i_2} \geq (i_1 - i_2)Q$. Therefore, Eq. (8) holds.

Since the calculation of $N_H(\mathbf{a}, A)$ has polynomial complexity, the complexity of the encoding/decoding procedure is polynomial in M and L'.

V. Conclusions and Future Work

This paper studies coding for channels motivated by DNA storage systems, in which data are encoded as a set of M unordered strings of length L. A K substitution error correcting code construction is presented for this channel. Our code achieves $O(K \log ML)$ redundancy for small K, which is order-wise optimal. The robust indexing technique we use in our code construction can be applied to deletion/insertion errors. It is interesting to find optimal codes that correct substitution or deletion/insertion errors for larger range of parameters K, M, and L.

References

[1] Z. Chang, J. Chrisnata, M. F. Ezerman, and H. M. Kiah, "Rates of DNA string profiles for practical values of read lengths," *IEEE Transactions on Information Theory*, vol. 63, no. 11, pp. 7166–7177, 2017.

[2] 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.

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

[4] Y. Erlich and D. Zielinski, "DNA fountain enables a robust and efficient storage architecture," *Science*, no. 6328, pp. 950–954, 2017.

[5] R. Gabrys, H. M. Kiah, and O. Milenkovic, "Asymmetric Lee distance codes for DNA-based storage," *IEEE Transactions on Information Theory*, vol. 63, no. 8, pp. 4982–4995, 2017.

[6] N. Goldman, P. Bertone, S. Chen, C. Dessimoz, E. M. LeProust, B. Sipos, and E. Birney, "Towards practical, high-capacity, low-maintenance information storage in synthesized DNA," *Nature*, no. 7435, pp. 77–80, 2013.

[7] R. Heckel, I. Shomorony, K. Ramchandran, and N. C. David, "Fundamental limits of DNA storage systems," *IEEE International Symposium on Information Theory* (ISIT), pp. 3130–3134, 2017.

[8] H. M. Kiah, G. J.Puleo, and O. Milenkovic, "Codes for DNA sequence profiles," *IEEE Transactions on Information Theory*, vol. 62, no. 6, pp. 3125–3146, 2016.

[9] D. E. Knuth, The Art of Computer Programming, Volume 4, Fascicle 3: Generating All Combinations and Partitions (Art of Computer Programming), Addison-Wesley Professional, 2005.

[10] A. Lenz, P. H. Siegel, A. Wachter-Zeh, and E. Yaakobi, "Coding over Sets for DNA Storage," *IEEE International Symposium on Information Theory* (ISIT), Vail, USA, 2018

[11] A. Lenz, P. H. Siegel, A. Wachter-Zeh, and E. Yaakobi, "Anchor-Based Correction of Substitutions in Indexed Sets," *IEEE International Symposium on Information Theory* (ISIT), Paris, France, 2019

[12] L. Organick, S. D. Ang, Y. J. Chen, R. Lopez, S. Yekhanin, K. Makarychev, M. Z. Racz, G. Kamath, P. Gopalan, B. Nguyen, C. Takahashi, S. Newman, H. Y. Parker, C. Rashtchian, G. G. K. Stewart, R. Carlson, J. Mulligan, D. Carmean, G. Seelig, L. Ceze, and K. Strauss, "Scaling up DNA data storage and random access retrieval," *bioRxiv*, 2017.

[13] N. Raviv, M. Schwartz, and E. Yaakobi, "Rank-Modulation Codes for DNA Storage with Shotgun Sequencing," *IEEE Transactions on Information Theory*, vol. 65, no. 1, pp. 50–64, 2018.

[14] R. Roth, Introduction to coding theory, Cambridge University Press, 2006.

[15] T. Shinkar, E. Yaakobi, A. Lenz, and A. Wachter-Zeh, "Clustering-Correcting Codes," *IEEE International Symposium on Information Theory* (ISIT), Paris, France, 2019

[16] I. Shomorony and R. Heckel, "Capacity results for the noisy shuffling channel," *IEEE International Symposium on Information Theory* (ISIT), Paris, France, 2019

[17] J. Sima, N. Raviv and J. Bruck, "On coding over sliced information," *IEEE International Symposium on Information Theory* (ISIT), Paris, France, 2019.

[18] J. Sima and J. Bruck, "Optimal k-deletion correcting codes," *IEEE International Symposium on Information Theory* (ISIT), Paris, France, 2019.

[19] W. Song, K. Cai, and K. A. S. Immink, "Sequence-Subset Distance and Coding for Error Control for DNA-based Data Storage ," *IEEE International Symposium on Information Theory* (ISIT), Paris, France, 2019

[20] S. M. H. T. Yazdi, Y. Yuan, J. Ma, H. Zhao, and O. Milenkovic, "A rewritable, random-access DNA-based storage system," *Scientific reports*, vol. 5, p. 14138, 2015.