New Levenshtein-Marker Code for DNA-based Data Storage Capable of Correcting Multiple Edit Errors

Zihui Yan and Cong Liang

Abstract

With the development of DNA synthesis and sequencing technologies, DNA becomes a promising medium for long-term data storage. Three types of errors may occur in the DNA strand, insertions, deletions and substitutions, which we collectively call edit errors. It is still challenging to design a code that can correct multiple edit errors on non-binary alphabets. In this paper, we propose a new coding schema for correcting multiple edit errors on DNA strands by splitting the whole strand into consecutive blocks with appropriate length and correcting a single edit error in each block. Our method, called the *DNA-LM* code, could be considered a generalization of the Levenshtein code combined with the marker code. We provide a linear encoding and decoding algorithm for our *DNA-LM* code. Compared to other encoding methods for DNA strands of several hundred base-pairs, our *DNA-LM* code achieved similar code rates and a much lower average nucleotide error rate in decoding.

Index Terms

DNA storage channel, Error-correcting codes, Synchronization errors, Varshamovs-Tenengolts codes, Marker codes.

I. Introduction

DNA is a promising storage medium with its longevity and enormous information density. In recent years, the researches and applications of DNA-based storage have been widely concerned and studied [1], [2], [3], [4], [5]. In a typical DNA storage pipeline (Fig. 1), digital messages were encoded into a massive amount of short DNA strands. The lengths of the DNA strings are usually 200-300 nucleotides (nt), which is limited by the current DNA synthesis technology. The encoded DNA strings were subsequently synthesized into DNA strands (also called DNA fragments), which could then be stored, duplicated, and

Z. Yan and C. Liang are with the Center for Applied Mathematics, Tianjin University, China, Tianjin, 300192 USA e-mail: cong.liang@tju.edu.cn.

read. The data extraction from the stored DNA contains DNA amplification via polymerase chain reaction (PCR) experiments and DNA sequencing. The DNA sequencing procedure includes random sampling and sequencing from a large pool of DNA fragments, which could be described by a random shuffling-sampling channel [6], [7]. DNA fragments were sequenced to varied depths, i.e., obtained a different number of reads. Undesired fragment structures, such as unbalanced GC content or long homopolymers, may also lead to no-readout for some DNA sequences, called dropouts [4], [8]. Besides the random sampling channel in the sequencing procedure, the DNA fragments themselves may obtain substitution, insertion, or deletion errors during the synthesize, amplification, and sequencing procedure [9]. We usually call an insertion or deletion error an indel, and call a substitution, insertion or deletion error an edit. Thus the DNA-based storage channel could be considered as a concatenated shuffling-sampling and edit-error channel [6], [10], [11].

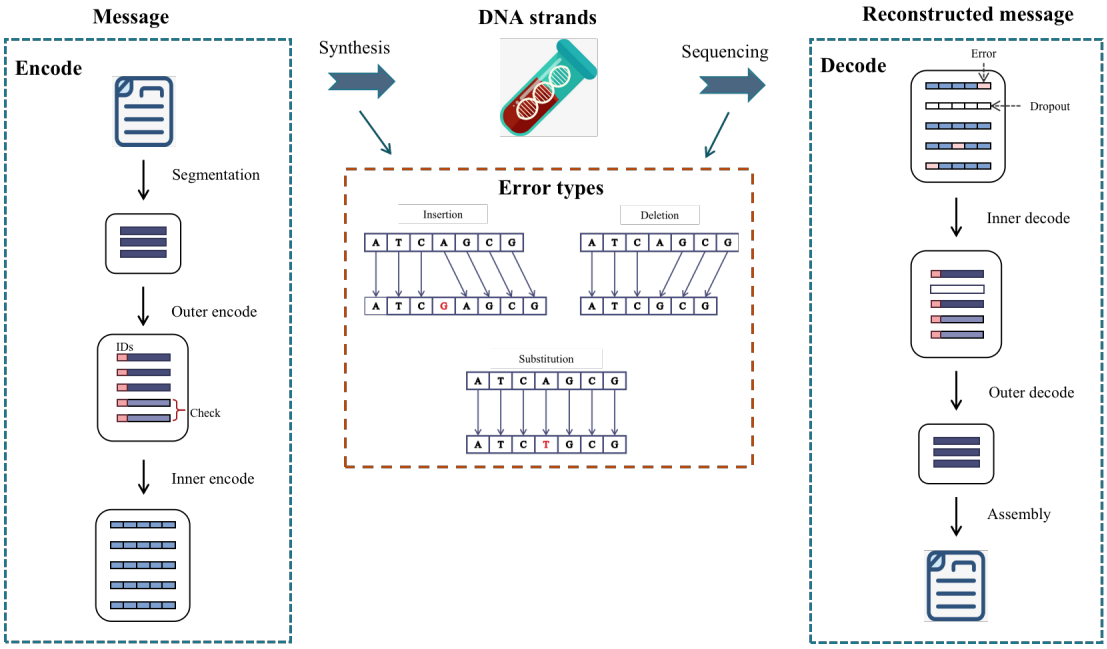


Fig. 1. Typical DNA-based storage system. Binary messages will be encoded and synthesized into DNA strands. DNA strands need to be sequenced and decoded to recover the original message. The DNA synthesis and sequencing procedures may bring additive errors and synchronization errors, which require error-correcting codes to ensure the accurate extraction of the stored information.

Various error correction methods have been employed in the DNA-based storage error channels. The most common design is a concatenated code, where the outer code deals with the dropouts in the shuffling-sampling channel, and the inner code deals with the edits in the DNA strands (Fig. 2). Some typical outer codes are the Reed-Solomon (RS) code [12], [13], [14], the low-density parity-check (LDPC) code [15], [16], [17], and the fountain code [4], [18], [19]. All of them have shown good performances in solving the

problem of dropouts of DNA strands. A variety of error correction codes have been proposed as the inner code, for example, the RS code [4], the HEDGES convolutional code [14], or the modified Levenshtein code by Cai [20]. Although the concatenated encoding scheme can usually achieve the desired error rate, none of the inner codes are optimal. The RS code can only correct substitution errors. Decoding failure would happen for any sequencing reads with indels. But studies have shown that the probabilities for insertions or deletions are at the same scale with substitutions [5]. The HEDGES is a convolutional code that is capable of correcting both indel and substitution errors. But the code has a low code rate and a high decoding complexity. Recently, Cai designed a single-edit-correction code with a high asymptotic code rate with linear encoding and decoding complexity. This code is attractive when the error rates per nucleotide are relatively low. While when the error rates increase, it is very likely that multiple errors exist in a single DNA strand, which Cai's code cannot correct. As a result, it is desired to design an inner code capable of correcting multiple edit errors with efficient encoding and decoding algorithms.

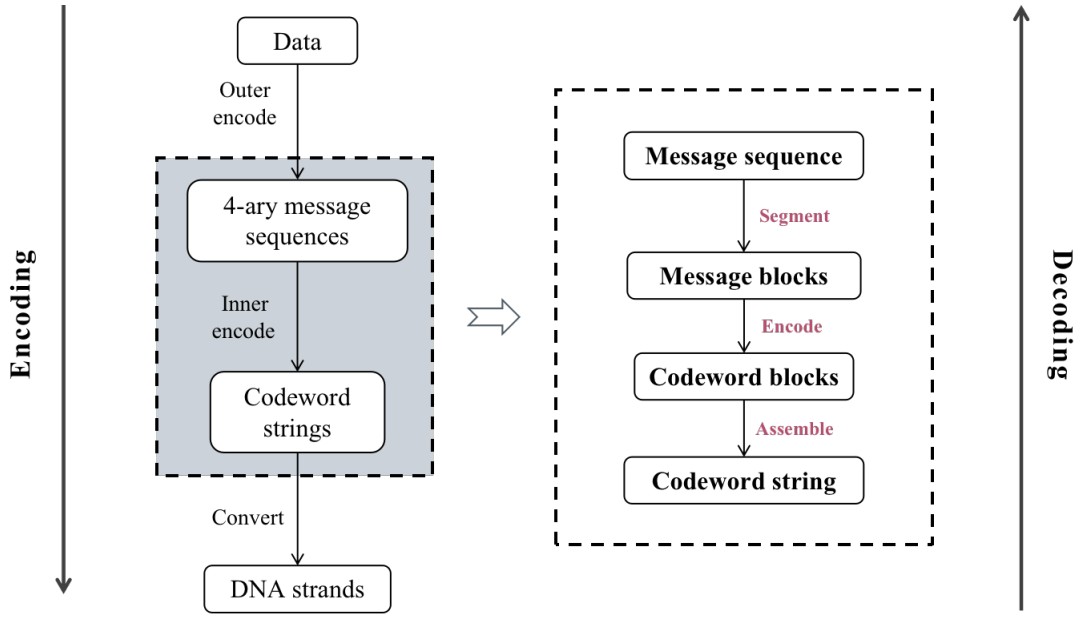


Fig. 2. Overview of the concatenated coding scheme and the inner code structure used in this paper.

The indel error in DNA sequences is analogous to the synchronization error in a communication channel [14]. Although compelling error-correction methods have been developed for channels such as the binary symmetric channel (BSC) or the binary symmetric erasure channel (BSEC), codes that could correct synchronization errors are still quite challenging to design [10], [21]. The Varshamovs-Tenengolts (VT) code, also called the Levenshtein code, is one of the most celebrated classes for correcting synchronization errors. VT codes that could correct single [22] or multiple [23] indels on binary alphabet

have been designed, and also be extended to non-binary alphabets [24], [25], [26], with an open-source implementation available at [27]. Many generalized Levenshtein methods have been applied to the DNA storage system [20], [28], [29], [30], [31]. But it remains difficult to correct multiple edit errors in the quaternary alphabet for DNA storage applications. Another class of codes for detecting synchronization errors is the marker codes [32]. They were first proposed in 1962 by Sellers, but haven't attracted much attention probably because they can only detect out-of-sync events but have no error correction capability between the markers. In this paper, we designed a hybrid Levenshtein-Marker code as the inner code for DNA-based storage (Fig. 2). The code consists of a series of composite codeword blocks. In each block, the code integrates the marker with the VT code to enable the detection and correction of one edit error. The error analysis of experiment of Organick et al. showed that the average error rate per position is 0.6%, with 0.4% substitutions, 0.2% deletions and 0.04% insertions [5]. For 200nt to 300nt length DNA strands, it is likely that more than one errors occurred in the whole strand, but we could segment the message into multiple blocks such that each block contains one error with high probability. Hence, our code has higher error correction capability than the single edit error-correcting VT codes in the quaternary alphabet.

Our paper is organized as follows. In section II, we summarize single edit-correcting VT codes on the quaternary alphabet. These are codes that could be utilized as the inner code in DNA-based storage systems, but none are not optimal. In section III, we introduce the Sellers marker code, and describe its concatenation with the codes in section II to generate multiple error-correcting codes. In section IV and section V, we present our new DNA Levenshtein-Marker (*DNA-LM*) code, and prove its error correction capability. We also provide the linear encoder and decoder algorithms of our code. In section VI, we discuss the capacities of this code, which include code rate, algorithm complexity and decoding success rate.

II. Single Error Correcting Levenshtein Codes on the Quaternary Alphabet

The Varshamov-Tenengolts (VT) codes [33] is a class of binary algebraic block codes which consists all binary vectors of length n belonging to

$$VT_{a,m}(n) = \left\{ \boldsymbol{x} \in \{0,1\}^n : \sum_{i=1}^{n} i x_i \equiv a \pmod{m} \right\}. \tag{1}$$

a is an integer with $0 \leq a \leq m-1$. It is usually called the syndrome or the remainder of sequence $\boldsymbol{x}$. Different a partition all length-n sequences $\boldsymbol{x} \in \{0,1\}^n$ into m sets. VT codes were first introduced for channels with asymmetric errors by Varshamov and Tenengolts [33]. Levenshtein [22] later proved that for $m \geq n+1$ and a fixed integer a with $0 \leq a \leq m-1$, VT codes are asymptotically optimal single-synchronization error correction codes. Levenshtein also showed that when $m \geq 2n$, VT codes can correct one insertion, deletion or substitution error. So VT codes were also referred as Levenshtein codes.

Although an efficient decoding algorithm for VT codes was known since 1965, a systematic encoding method for VT code that can correct a single insertion or deletion was introduced only until 1998 by Abdel-Gaffar et al [34]. Later, Saowapa at el [35] adopted it to get a systematic encoder for codes that can correct a single edit. In brief, the idea is to insert "parity" bits at dyadic positions to ensure that the constructed codeword has the desired syndrome. We call this **Encoder *SS*** (systematic encoded by Saowapa), and restate the linear encoding method here since we will utilize this method for the interleaved encoding scheme in Section II - B.

Encoder *SS*: For any message sequence $\boldsymbol{x} = \{x_1, x_2, \ldots, x_k\} \in \{0,1\}^k$, the encoder SS can stick it into a Levenshtein codeword $\boldsymbol{y} = SS(\boldsymbol{x}) \in VT_{a,2n}(n)$, where $k = n - \lceil \log n \rceil - 1$. The encoding idea is to insert "parity" bits at dyadic positions, ie., c_{2^i}, for $0 \leq i \leq t-2$ and c_n, and attach message symbols to other positions, to ensure that $\sum_{i=1}^{n} i y_i \equiv a \pmod{2n}$. Here $t = n - k$ is the number of redundancy bits.

Example: Consider the message is 01011 and $a = 0$, so $n = 10$, $t = 5$ and $m = 20$. The codeword $\boldsymbol{y} = (y_1, y_2, \ldots, y_{10})$ should satisfy that $\sum_{i=1}^{10} i y_i \equiv \sum_{j=1}^{4} 2^{j-1} y_{2^{j-1}} + 10 \cdot y_{10} + 5 + 7 + 9 \equiv 0 \pmod{20}$. And expand $19 - 10 = 9$ into binary form $1 \cdot 2^3 + 1 \cdot 2^0$. Hence codeword is $\bar{1}\bar{0}0\bar{0}101\bar{1}1\bar{1}$, where the overbar bits are check bits.

Many scholars have studied the generalization of VT code to non-binary alphabets [24], [25], [26]. Since the DNA base system contains four elements, in this work, we focus on codes that could correct edit errors for the quaternary alphabet. We map four DNA nucleotides $\mathcal{D} = \{A, T, G, C\}$ to the quaternary alphabet $\Sigma = \{0, 1, 2, 3\}$:

$$A \leftrightarrow 0, \quad T \leftrightarrow 1, \quad C \leftrightarrow 2, \quad G \leftrightarrow 3. \tag{2}$$

Given any DNA sequence $\boldsymbol{\sigma} \in \mathcal{D}^N$, it could be one-to-one mapped to a quaternary sequence $\boldsymbol{x} \in \Sigma^N$. As a result, we discuss the encoding method from a binary sequence to a quaternary sequence on Σ. The encoded quaternary sequence on Σ could then be mapped to DNA sequences following the rule above. Next, we describe some quaternary codes that could correct a single indel or edit error.

A. Non-binary VT codes for single indel correction

Tenengolts generalized Levenshtein's single indel-correcting codes to non-binary alphabets in 1984 [24]. For any q-ary sequence $\boldsymbol{s} = (s_1, s_2, \ldots, s_n)$, Tenengolts defined a corresponding length $(n-1)$ auxiliary binary sequence $A_{\boldsymbol{s}} = (\alpha_1, \alpha_2, \ldots, \alpha_{n-1})$, where $\alpha_i = 0$ when $s_i < s_{i-1}$ and otherwise $\alpha_i = 1$. And defined a "parity" check function

$$sum(\boldsymbol{s}) \equiv \sum_{i=1}^{n} s_i \pmod{q}.$$

For $0 \le a \le n-1$ and $0 \le b < q$, the Tenengolts's code is defined as

$$Ten_{a,b}(n) := \left\{ \boldsymbol{s} \in \mathbb{Z}_q^n : sum(\boldsymbol{s}) = b, \sum_{i=1}^{n-1} i\alpha_i \equiv a \pmod{n} \right\}.$$

Each of the sets $Ten_{a,b}(n)$ is a single indel error correcting code.

Tenengolts's paper introduced the algebraic structure of the code, but didn't provide a method to encode messages into such codes. Only recently, Abroshan et al [25] proposed a method to systematically map k bits message onto n length q-ary generalized Levenshtein codeword. We call it **Encoder *TA*** (introduced by Tenengolts and systematic encoded by Abroshan). Considering its application in DNA storage, we take $q = 4$, and have

$$k = 2n - 3\lceil \log n \rceil - 2.$$

B. Interleaved binary VT codes

Since we are particularly interested in codes for 4-ary alphabets, it is also quite intuitive to construct a quaternary code by interleaving two binary Levenshtein codes. The quaternary code interleaved by two binary one-edit-correction Levenshtein codes is capable of correcting one edit error. This idea was first mentioned in Helberg [23], but hasn't been explicitly stated in the literature. Here we call this **Encoder *IBS*** (interleaved binary VT codes and encoded by Saowapa) and describe the method in detail.

Definition 1. *For a sequence $\boldsymbol{y} = (y_1, y_2, \ldots, y_k) \in \Sigma^k$, let $y_i = y_i^{(0)}2^0 + y_i^{(1)}2^1$ be the binary form of symbol y_i, then $\boldsymbol{y^{(0)}} = (y_1^{(0)}, \ldots, y_n^{(0)})$, $\boldsymbol{y^{(1)}} = (y_1^{(1)}, \ldots, y_n^{(1)})$. $\boldsymbol{y}$ is the interleaved sequence of $\boldsymbol{y^{(1)}}$ and $\boldsymbol{y^{(2)}}$, which we denote by $\boldsymbol{y^{(1)}} \parallel \boldsymbol{y^{(0)}}$.*

Encoder *IBS*: Then for $2k$ bits message set, we can encode them to the codeword set

$$IBS_{k,a}(n) = \left\{ \boldsymbol{y} = \boldsymbol{y^{(1)}} \parallel \boldsymbol{y^{(0)}} : \boldsymbol{x} = \boldsymbol{x^{(1)}} \parallel \boldsymbol{x^{(0)}} \in \Sigma^k, \boldsymbol{y^{(i)}} = \textbf{Encoder } \boldsymbol{SS}(\boldsymbol{x^{(i)}}) \in VT_{a,2n}(n), i = 0, 1 \right\},$$

Here $k = n - \lceil \log_2(n) \rceil - 1$, and we can conclude that $IBS_{k,a}(n)$ is a single edit error-correcting code on the quaternary alphabt.

Example: Consider the message is 02312 and can be expanded as 01101 || 00110. Set $a = 0$, so $n = 10$ and $m = 20$. Encode $SS(01101) = 0000110010$ and $SS(00110) = 1101011000$. Hence the codeword $\boldsymbol{y} = 0000110010 \parallel 1101011000 = 1101231020$.

C. Cai order-optimal code

The redundancy of VT codes and their generalizations is always $K \log n + o(\log n)$ bits, where n is the length of the codeword, and K is a constant. Small K is usually preferred to ensure high code rate. To minimize K, Cai [20] designed a single edit correcting code on the quaternary alphabet by enforcing the k-sum balanced constraint on the message sequence and appending the syndromes next to it. Here we call it **Encoder *Cai***. The redundancy of it is $\lceil \log n \rceil + O(\log \log n)$ bits. It has $K = 1$ and Cai called it order-optimal. Specifically, this code encodes $2k$ bits message and outputs a length n quaternary codeword, where $n = (k + \lceil (\log(4k+5))/2 \rceil + \lceil (\log 1440 + \log\log(k+1))/2 \rceil + 6)$. As mentioned in Cai's paper, this code has an efficient code rate when n is large (say $n > 512$). While the advantage of order-optimality is impacted when the message sequence is short because some constant redundant bits are required.

III. Multiple Indel Correction and the Marker Code

The codes mentioned above all have linear encoding and decoding methods, but their application is still limited since they could only correct a single indel or edit error. To solve this problem, Helberg [23] introduced a class of binary codes that can correct multiple edit errors by increasing the intervals between each congruent class. Tuan and Hieu [26] generalized this method to non-binary alphabets. However, no encoding method has been proposed for these codes, and their codeword contains much more redundancy bits compared to the original design of the VT codes.

Another technique to deal with synchronization errors is the marker code. Sellers [32] first suggested inserting the subsequence "001" into a long sequence at regular intervals as "markers" to detect synchronization errors. Because the markers appear periodically in the transmitted sequence, the synchronization can be regained by aligning the markers. If there is a shift of the marker position, deletion or insertion events could be detected. However, the marker code has no error correction capability: we could regain synchronization in future sequences according to the marker position, but we cannot correct the fragment that contains synchronization errors, not to mention correct substitution errors.

To tackle the issue that the marker code has no error correction capability, we could intuitively encode the messages between markers with the single-edit-correction codes that we have described in section II. Ferreira et al [36] has proposed a similar idea, where they used markers with Levenshtein encoded blocks to detect and correct edit errors. In the scenario of DNA storage, current synthesis technologies can produce strands of lengths 200nt to 300nt. We can split the message sequence into several shorter blocks, then encode each message block into a codeword block, and insert markers between blocks. This method enables us to correct more than one error in the whole DNA strand. We illustrate the strand structure in Fig. 3. We call the concatenated codes with such a construction the ***4-ary TA-Marker*** code, the ***IBS-Marker*** code, and the ***Cai-Marker*** code, respectively. Note that the code rate of the whole strand will decrease a bit because of the marker.

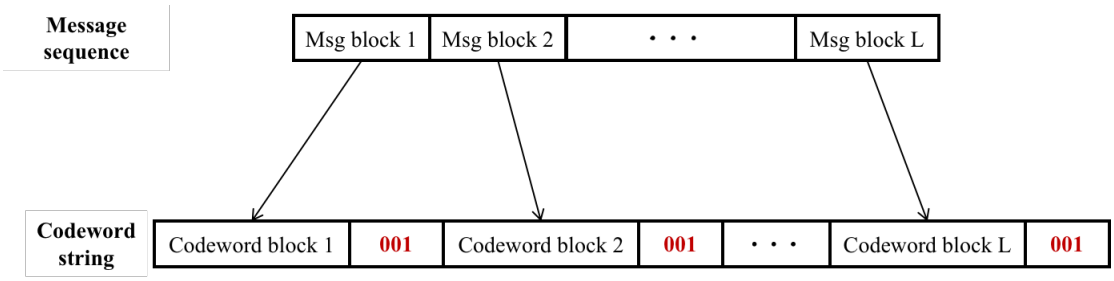


Fig. 3. The concatenated encoding structure of the generalized Levenshtein codes and the marker code proposed in this paper.

With the design above, we can correct multiple errors in each DNA strand. But if an error occurred in the marker region, the decoder will not be able to identify the marker and the synchronization will be lost, then the codewords before and after the marker will be corrupt. This is because the segmentation of codewords relies on the correct identification of the marker position, but no protective encoding was designed for the marker code. As a result, we redesigned the marker codes and integrated them with a modified systematic encoding of VT codes. Compared to the single error correction codes in section II, our hybrid code achieved a higher error-correcting rate while maintaining a high code rate for DNA sequences between 200-300 base pairs. We describe this code in the next section.

IV. The generalized Levenshtein-Marker code

We are also interested in the application of systems of congruences like (1) in the encoding for DNA storage. We designed a class of 4-ary codes that can correct multiple edit errors, and termed it the DNA Levenshtein-Marker (***DNA-LM***) code. The *DNA-LM* code is concatenated by several segments, which we

call "blocks" in this paper. Each of the blocks is encoded with a systematic DNA-segment-Levenshtein-Marker (***DNA-sLM***) code. The *DNA-sLM* code consists of five components, the marker, the message, the check, the separator, and the syndrome. We designed the separator and the marker to locate the positions of the message and the syndrome cooperatively. The message and the syndrome can correct each other. Knowing the sequence of one could assist the identification of the other. The check was designed to locate the error if a substitution event happens. We show below that the *DNA-sLM* code is a single edit correction code, so one error in the marker segment will not deterministically corrupt the entire block. To encode a message sequence into a *DNA-LM* codeword string, we first split the message sequence into several short blocks, then encode each message block into a *DNA-sLM* codeword, and concatenate them together (Fig. 4). Note that the first block do not need the marker to keep synchronization so that we skip it and encode the message straightly.

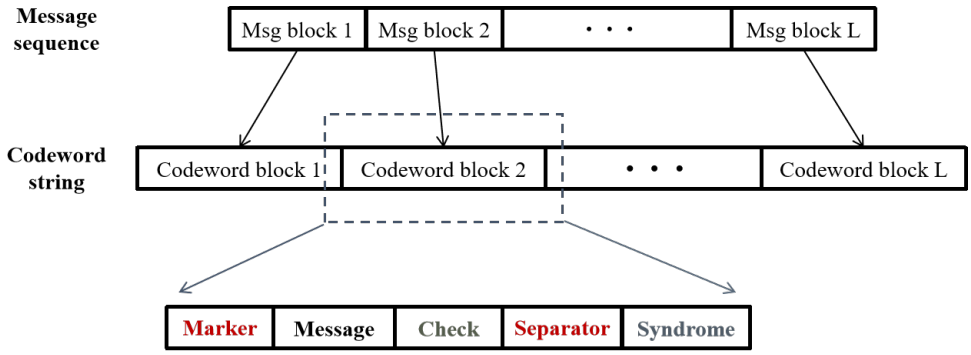


Fig. 4. The construction of a DNA Levenshtein-Marker codeword.

We introduce some relevant notations and terminologies to describe our *DNA-sLM* code in detail. Denote the message sequence $\boldsymbol{x} = (x_1, x_2, \ldots, x_k) \in \Sigma^k$.

- $Syn : \{0,1\}^k \to [0, 2k)$ calculates the syndrome of a binary sequence. For a binary sequence $\boldsymbol{z} \in \{0,1\}^k$, $Syn(\boldsymbol{z})$ is a function that

$$Syn(\boldsymbol{z}) \equiv \sum_{i=1}^{k} i z_i \pmod{2k}. \tag{3}$$

  We use $BSyn(\boldsymbol{z})$ to represent the binary form of the number $Syn(\boldsymbol{z})$. Recall the definition 1, for quaternary message sequence $\boldsymbol{x} = \boldsymbol{x^{(1)}} \parallel \boldsymbol{x^{(0)}}$, where $\boldsymbol{x^{(1)}}, \boldsymbol{x^{(0)}} \in \{0,1\}^k$, the map $IBSyn : \Sigma^k \to \Sigma^t$ satisfies

$$IBSyn(\boldsymbol{x}) = BSyn(\boldsymbol{x^{(1)}}) \parallel BSyn(\boldsymbol{x^{(0)}}),$$

here, $t = \lceil \log(2k) \rceil$. We call $IBSyn(\boldsymbol{x})$ the syndrome sequence of $\boldsymbol{x}$.

- $Ck : \Sigma^k \to \Sigma$ is the check function of the message $\boldsymbol{x}$, where $Ck(\boldsymbol{x}) \equiv \sum_{i=1}^{k} x_i \pmod 4$.
- $Sp : \Sigma^3 \to \Sigma$ is the separator function, where $Sp(a, b, c)$ is the symbol which not in the set $\{a, b, c\}$. To be specific, if $a+2 \neq b, c$, set $Sp(a, b, c) = a+2$, if $a+3 \neq b, c$, set $Sp(a, b, c) = a+3$, otherwise, $Sp(a, b, c) = a + 1$.
- $Mk : \Sigma^3 \to \Sigma$ is the marker function, which is same as the function Sp.

Definition 2. *The DNA-sLM code sLM(k) is defined as*

$$\begin{aligned} sLM(k) := \Big\{ & (m, m, x_1, x_2, \ldots, x_k, c, s, s, s, a_1, \ldots, a_t) \in \Sigma^{k+t+6} : \\ & c = Ck(x_1, x_2, \ldots, x_k), (a_1, a_2, \ldots, a_t) = IBSyn(x_1, x_2, \ldots, x_k), \\ & s = Sp(c_1, a_1, a_2), m = Mk(f, x_1, x_2) \Big\}. \end{aligned}$$

Here f is the last symbol of the previous codeword block. And we call (m, m) the marker segment, $\boldsymbol{x} = (x_1, x_2, \ldots, x_k)$ the message segment, (c) the check segment, (s, s, s) the separator segment, and $\boldsymbol{a} = (a_1, a_2, \ldots, a_t)$ the syndrome segment. These five segments form one codeword block of the DNA-LM codeword string.

Since we are interested in concatenating multiple *DNA-sLM* codes into one consecutive codeword string, in the rest of this section, we discuss the error correction capability of the *DNA-sLM* code assuming that the received codeword has an undetermined endpoint. Theorem 1 states that the first message block $\boldsymbol{x}$ as well as the starting position of the next codeword block could be uniquely determined from the received codeword string $\boldsymbol{r}$, when there is at most one edit error in the first block. Thus after decoding the first block, we can trim the codeword string at the starting position of the next block and iterate the decoding process.

Theorem 1. *The DNA-sLM code sLM(k) can correct a single edit error, and realize re-synchronization after this block even if the decoder doesn't know where the block ends.*

Before proving this theorem, we need to generate some lemmas.

Let $\overline{\boldsymbol{x}}_{sub}$, $\overline{\boldsymbol{x}}_{ins}$, and $\overline{\boldsymbol{x}}_{del}$ be the first k elements of a sequence obtained from $\boldsymbol{x}$ via a single edit when there is a substitution, insertion, or deletion, respectively. For $\forall r \in \Sigma$, the obtained sequence has the form $\overline{\boldsymbol{x}}_{sub} = (x_1, \ldots, x_{j-1}, r, x_{j+1}, \ldots, x_k)$, $\overline{\boldsymbol{x}}_{ins} = (x_1, \ldots, x_{j-1}, r, x_j, \ldots, x_{k-1})$, or $\overline{\boldsymbol{x}}_{del} = (x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_k, r)$. We refer to them collectively as $\overline{\boldsymbol{x}}$. Note that if a deletion happens, a symbol was appended at the end

of the sequence. Here we consider the case of the *DNA-sLM* code, if a deletion happens in the message, the check $c = Ck(\boldsymbol{x})$ will be added to the end of it. The following lemmas consider the result returned by the check and syndrome function as $\boldsymbol{x}$ changes to $\overline{\boldsymbol{x}}$.

Lemma 1. *For any sequence $\overline{\boldsymbol{x}}_{sub}$ obtained via a single substitution from $\boldsymbol{x}$, $Ck(\overline{\boldsymbol{x}}_{sub})$ will not be equal to $Ck(\boldsymbol{x})$.*

Proof. Let $\overline{c} = Ck(\overline{\boldsymbol{x}}_{sub})$ and $c = Ck(\boldsymbol{x})$,

$$\begin{aligned} c - \overline{c} &\equiv \sum_{i=1}^{k} x_i - \left(\sum_{i=1}^{j-1} x_i + r + \sum_{i=j+1}^{k} x_i \right) \pmod 4 \\ &\equiv x_j - r \pmod 4. \end{aligned}$$

For $x_j \neq r$, c will never equal $\overline{c}$. □

Hence, when the length of the message segment is correct, if the message does not match the syndrome, the check can help the decoder detect where the substitution error is. ***

Lemma 2. *For any sequence $\overline{\boldsymbol{x}}$ obtained from $\boldsymbol{x}$, $IBSyn(\overline{\boldsymbol{x}})$ will not be equal to $\boldsymbol{a}$.*

Proof. We first discuss when there is an insertion in $\boldsymbol{x}$. We have $\overline{\boldsymbol{x}}_{ins} = (x_1, \ldots, x_{j-1}, r, x_j, \ldots, x_{k-1})$. We calculate the syndrome $\overline{a^{(0)}}$ and $\overline{a^{(1)}}$ for $\overline{\boldsymbol{x}}_{ins}$:

$$\begin{aligned} \sum_{i=1}^{j-1} i x_i^{(0)} + j r^{(0)} + \sum_{i=j}^{k-1} (i+1) x_i^{(0)} &\equiv \overline{a^{(0)}} \pmod{2k}; \\ \sum_{i=1}^{j-1} i x_i^{(1)} + j r^{(1)} + \sum_{i=j}^{k-1} (i+1) x_i^{(1)} &\equiv \overline{a^{(1)}} \pmod{2k}. \end{aligned}$$

We can see that

$$\begin{aligned} a^{(s)} - \overline{a^{(s)}} &\equiv \sum_{i=1}^{k} i x_i^{(s)} - \Big(\sum_{i=1}^{j-1} i x_i^{(s)} + j r^{(s)} + \sum_{i=j+1}^{k-1} (i+1) x_i^{(s)} \Big) \pmod{2k} \\ &\equiv k x_k^{(s)} - \sum_{i=j}^{k-1} x_i^{(s)} - j r^{(s)} \pmod{2k} \end{aligned}$$

Therefore, for $s = 0, 1$, $a^{(s)} - \overline{a^{(s)}} \equiv 0 \pmod{2k}$ if and only if $(r^{(s)}, x_j^{(s)}, \ldots, x_{k-1}^{(s)})$ are all zeros or ones. These cases are equivalent to no symbol that has been changed in the sequence. The same could be proved when there is one deletion or substitution error.

Because the received sequence is interleaved by two binary sequences, if one symbol changed in transit, at least one of these two binary sequences will be changed. So for any received sequences $\overline{\boldsymbol{x}}$, as long as $\overline{\boldsymbol{x}} \neq \boldsymbol{x}$, the syndrome sequence $IBSyn(\overline{\boldsymbol{x}}) \neq IBSyn(\boldsymbol{x})$. □

From lemma 2, we can see that even if we can not figure out the boundary of the message segment, we won't either receive the error transmitted codeword as the correct one. Therefore, as long as one of the message part and the syndrome part transmit correctly, we can decode the received. Besides, because we only consider one edit error situation, if there is one substitution error in message segment, from lemma 1, the check function will not be satisfied as well, so that the check can help the decoder to find whether the error is in the message segment or syndrome segment.

We next consider the truncated code without the marker segment, which we denote as ***t-sLM***(k). We show its error correction capability in Lemma 3.

Lemma 3. *The code t-sLM(k) can correct a single edit error without the information of the received block length.*

Proof. For any message $\boldsymbol{x}$, denote its corresponding codeword $\boldsymbol{y} = (\boldsymbol{x}, c, s, s, s, \boldsymbol{a})$. We consider the case that the end position of the received codeword is unknown. Let the single edit error domain of codeword $\boldsymbol{y}$ be

$$\mathcal{D}(\boldsymbol{y}) := \Big\{\boldsymbol{r} : \boldsymbol{r} \text{ is obtained from } \boldsymbol{y} \text{ by one edit error}\Big\} \cup \Big\{\boldsymbol{y}\Big\}. \tag{4}$$

Let $\boldsymbol{r}_k$ denote the first k elements of $\boldsymbol{r}$. Take $\overline{\boldsymbol{c}} = Ck(\boldsymbol{r}_k)$, and $\overline{\boldsymbol{a}} = IBSyn(\boldsymbol{r}_k)$. We first define some subsets of $\mathcal{D}(\boldsymbol{y})$ to represent the different error types and error locations.

- Let Ck denote the event that the received sequence is in the following form:

$$Ck := \big\{\boldsymbol{r} : r_{k+1} = Ck(\boldsymbol{r}_k)\big\}. \tag{5}$$

  We can see that Ck is the subset of $\mathcal{D}(\boldsymbol{y})$, and if the message and the check are correct in transmit, the received $\boldsymbol{r}$ should belong to Ck.

- Let Sp_{-1}, Sp_0 and Sp_{+1} denote the events that the received is in following forms:

$$Sp_0 := \big\{\boldsymbol{r} : r_{k+2} = r_{k+3} = r_{k+4}\big\};$$

$$Sp_{+1} := \big\{\boldsymbol{r} : r_{k+3} = r_{k+4} = r_{k+5} \neq r_{k+2}\big\};$$

$$Sp_{-1} := \big\{\boldsymbol{r} : r_{k+1} = r_{k+2} = r_{k+3} \neq r_{k+4}\big\}.$$

Sp_{-1}, Sp_0 and Sp_{+1} are the subsets of $\mathcal{D}(\boldsymbol{y})$ as well. Suppose that the separator is correct, then if there is no indel in the message and the check, $\boldsymbol{r} \in Sp_0$; if there is one deletion in the message and the check, $\boldsymbol{r} \in Sp_{-1}$; and if there is one insertion in the message and the check, $\boldsymbol{r} \in Sp_{+1}$. Besides, due to the structure of the separator, Sp_{-1}, Sp_0 and Sp_{+1} are mutually disjoint. If the separator is not correct, the received will not belong to any sets above. Hence, we use $Sp = Sp_{-1} \cup Sp_0 \cup Sp_{+1}$ to represent the event that the separator segment is correct in transmit.

- Let Syn_{-1}, Syn_0 and Syn_{+1} denote the events that the received is in the following forms:

$$Syn_i = \left\{\boldsymbol{r} : (r_{k+5+i}, r_{k+6+i}, \ldots, r_{k+4+t+i}) = IBSyn(\boldsymbol{r}_k)\right\} \cap (Sp_i \cup Sp^c) .$$

  Here $i \in \{-1, 0, +1\}$. We use $i = +1$, -1, and 0 to indicate the changes of the syndrome position. If the message and the syndrome are correct in transmit, $\boldsymbol{r}$ should belong to one of these sets. Let $Syn = Syn_{-1} \cup Syn_0 \cup Syn_{+1}$ represent the event that the syndrome and the message are matched.

According to the above definitions, we make several observations:

- If the whole string is correct in transmit, $\boldsymbol{r}$ should belong to set $Ck \cap Sp_0 \cap Syn_0$;
- Because we only discuss about single edit error, we can see that if the message is correct in transmit,

$$\boldsymbol{r} \in \big(Ck \cap Syn\big) \cup \big(Ck \cap Sp_0\big) \cup_{i\in\{-1,0,+1\}} \big(Sp_i \cap Syn_i\big). \tag{6}$$

  Based on our assumption, when the message is correct, only one function of the check, the separator, and the syndrome has not matched the message. Hence when two of them are matched, we determine that the message is correct. For example, $\boldsymbol{r} \in Sp_{-1}$ indicates the event that the separator segment was correct in transmit and was moved forward one symbol in transmit, which suggests that one error occurred before the separator segment. In this situation, the syndrome was moved forward one symbol as well. If not, it implies that the received sequence $\boldsymbol{r}$ occurred errors both in the separator and the syndrome which contradicts the assumption of one edit error. Hence, if the message is correct, then $IBSyn(\boldsymbol{x}) = (r_{k+4}, r_{k+5}, \ldots, r_{k+3+t})$, $\boldsymbol{r}$ should belong to Syn_{-1}. And this situation indicates that the check occurred one deletion in transmit.
- According to lemma 2, if one substitution occurred in the message, $IBSyn(\boldsymbol{r}_k) \neq (r_{k+5}, r_{k+6}, \ldots, r_{k+4+t})$, so that $\boldsymbol{r} \notin Syn_0$. And because the separator and the check is correct, according to lemma 1, $\boldsymbol{r} \in Ck^c \cap Sp_0$. Hence $\boldsymbol{r} \in Ck^c \cap Sp_0 \cap Syn_0^c$. And the same relations can be proved that if one insertion occurred in the message, $\boldsymbol{r} \in Sp_{+1} \cap Syn_{+1}^c$, and if one deletion occurred in the message,

$\boldsymbol{r} \in Sp_{-1} \cap Syn_{-1}^{c}$. Therefore, if the message is not correct,

$$\boldsymbol{r} \in \left(Ck^{c} \cap Sp_0 \cap Syn_0^{c}\right) \cup_{i\in\{-1,+1\}} \left(Sp_i \cap Syn_i^{c}\right). \tag{7}$$

- Donate the above sets by

$$S_1 = Ck \cap Sp_0 \cap Syn_0; \tag{8}$$

$$S_2 = \left(Ck \cap Syn\right) \cup \left(Ck \cap Sp_0\right) \cup_{i\in\{-1,0,+1\}} \left(Sp_i \cap Syn_i\right); \tag{9}$$

$$S_3 = \left(Ck^{c} \cap Sp_0 \cap Syn_0^{c}\right) \cup_{i\in\{-1,+1\}} \left(Sp_i \cap Syn_i^{c}\right). \tag{10}$$

We can figure out that $S_1 \subset S_2$, and $S_2 \cap S_3 = \emptyset$, $S_2 \cup S_3 = \mathcal{D}(\boldsymbol{y})$.

Suppose that there are two codewords $\boldsymbol{y_1}, \boldsymbol{y_2}$, and a received sequence $\boldsymbol{r}$ which simultaneously belongs to $\mathcal{D}(\boldsymbol{y_1})$ and $\mathcal{D}(\boldsymbol{y_2})$. Assume that the message segments of codewords $\boldsymbol{y_1}, \boldsymbol{y_2}$ are $\boldsymbol{x_1}, \boldsymbol{x_2}$ respectively. So $\boldsymbol{x_1} \neq \boldsymbol{x_2}$.

According to the above observations, if the received sequence $\boldsymbol{r}$ does not belong to set S_2, we know that the message is not correct. Hence, both the separator and the syndrome are correct. In this case, we could locate the position of the syndrome by checking which event of Sp_0, Sp_{-1}, and Sp_{+1} happens. Then the syndrome is the first t symbols after the separator, and we denote it by $\widehat{\boldsymbol{a}}$. Now we have two message segments $\boldsymbol{x_1}$, $\boldsymbol{x_2}$, with the same syndrome $IBSyn(\boldsymbol{x_1}) = IBSyn(\boldsymbol{x_2}) = \widehat{\boldsymbol{a}}$. Since the edit distance and the Hamming distance between two VT codewords are at least 3, we must have $d_{edit}(\boldsymbol{x_1}, \boldsymbol{x_2}) \geq 3$ and $d_{Hamming}(\boldsymbol{x_1}, \boldsymbol{x_2}) \geq 3$. At least one of these distances between $\boldsymbol{r}$ and $\boldsymbol{y_1}$, $\boldsymbol{y_2}$ is greater than 1. This is conflict to the assumption that there is only one edit in the received sequence $\boldsymbol{r}$. So $\boldsymbol{x_1} = \boldsymbol{x_2}$ and $\boldsymbol{y_1} = \boldsymbol{y_2}$. On the other hand, if $\boldsymbol{r}$ does not belong to the set S_3, the message in $\boldsymbol{r}$ is correct. Again, we have $\boldsymbol{x_1} = \boldsymbol{x_2}$, and $\boldsymbol{y_1} = \boldsymbol{y_2}$.

Above all, for any *t-sLM*(k) codewords $\boldsymbol{y_1} \neq \boldsymbol{y_2}$,

$$\mathcal{D}(\boldsymbol{y_1}) \cap \mathcal{D}(\boldsymbol{y_2}) = \emptyset, \tag{11}$$

so the code *t-sLM*(k) is a single edit error-correcting code. □

As a result, the error position could be uniquely determined by checking the received sequence belongs to sets S_2, S_3. Once we know the error position, we could easily recover $\boldsymbol{x}$ from $\boldsymbol{r}$. If the message is correct, where $\boldsymbol{r} \in S_2$, we take the first k elements of $\boldsymbol{r}$ as the message. Otherwise, if $\boldsymbol{r} \in S_3$, we identify

the error type in $\boldsymbol{x}$ as follows:

1) $\boldsymbol{r} \in Sp_0 \Leftrightarrow \boldsymbol{x}$ has one substitution;

2) $\boldsymbol{r} \in Sp_{+1} \Leftrightarrow \boldsymbol{x}$ has one insertion;

3) $\boldsymbol{r} \in Sp_{-1} \Leftrightarrow \boldsymbol{x}$ has one deletion.

Then we can locate the correct syndrome and use it to decode the message sequence.

From above, we can see that (s, s, s) altogether were designed to separate and locate the positions of the message sequence and the syndrome sequence. They are similar to the Seller's marker codes, which play an important role in identifying where the indel happened. Then the check bit c was designed to identify the error position if a substitution happens. Overall, Lemma 3 shows that if there is no error in the marker the *sLM(k)* code, we can correct one edit error in the truncated codeword *t-sLM*(k). With this, we show that *sLM(k)* is also a single-edit-correction code.

Lemma 4. *If one edit occurred in the received sLM(k) codeword, wherever it is in the marker or other segments, we can determine the correct message, and realize the re-synchronization after this block.*

Proof. Suppose that the decoder received the sequence $\boldsymbol{r}$, for consistency of the proof, let (r_{-1}, r_0) be the first two element of $\boldsymbol{r}$, which is presumably the marker position. If there is no error in the marker code, we can see that $r_{-1} = r_0$. While if an edit error occurred in the marker, because of the design of the marker function, we have $r_{-1} \neq r_0$,. As a result, we could determine whether an edit error occurred in the marker or not by checking the first two elements of the received. If $r_{-1} = r_0$, there is no error occurred in the marker. From lemma 3, we know that we can correct on edit error in the codeword *t-sLM*(k). If $r_0 \neq r_1$, there is one edit error in marker code. It remains to show that the same could be achieved in this case.

Following previous notations, let Ck_m denote the sequence sets that the sequences match that $Ck(r_{1+m}, r_{2+m}, \ldots, r_{k+}$ r_{k+1+m}. Sp_m and Syn_m are defined as in lemma 3, where $m = -1, 0, 1$. Table I lists the sets that the received sequence should belong to when different error types occur in the marker.

TABLE I

TABLE FOR ERROR TYPES IN THE RECEIVED SEQUENCE.

Error type	Marker	Check	Separator	Remainder
sub	(x,m) or (m,x)	Ck_0	Sp_0	Syn_0
del	m	Ck_{-1}	Sp_{-1}	Syn_{-1}
ins	(x,m,m) or (m,x,m)	Ck_{+1}	Sp_{+1}	Syn_{+1}

Here, x can be an arbitrary symbol in Σ except for m. So that if $r_{-1} \neq r_0$, the decoder will detect that there is one error in the maker code. In this case, we do not use the marker code to figure out the block beginning. On the contrary, we use three equations to detect the error type of the marker code to ensure synchronization.

Let the single edit error domain of codeword $\boldsymbol{y}$ be

$$\mathcal{D}_{marker}(\boldsymbol{y}) := \Big\{\boldsymbol{r} : \boldsymbol{r} \text{ is obtained from } \boldsymbol{y} \text{ by one edit error}, r_{-1} \neq r_0\Big\}. \tag{12}$$

Because the separator of any received sequence will be satisfied with only one form in the above table, and the $k+1$ to 2 symbols before the separator form the message, we can not make a mistake in decoding the message. And according to the marker function, the unique marker code can be picked up by the last symbol of the previous block and the first two symbols in the current block. Therefore, the sets $\mathcal{D}_{marker}(\boldsymbol{y})$ for different $\boldsymbol{y} \in$ *sLM(k)* are disjoint. □

Lemma 5. *If there is one edit error that occurred in the first block, we can re-synchronize the sequence after decoding this block.*

Proof. Firstly, consider the condition of no error in the syndrome $\boldsymbol{a}$. We can find the boundary of the syndrome, and the end of the syndrome is the beginning of the next block. Then we now discuss on conditions of the incorrect syndrome. Because the separator is correct, for brevity, let the received sequence which has been truncated after separator be $\boldsymbol{r}$, and the correct syndrome $\boldsymbol{a} = IBSyn(\boldsymbol{x})$. Let

$$\begin{aligned}
\mathcal{D}_1(\boldsymbol{m}) &= \Big\{(r_t, r_{t+1}, r_{t+2}, r_{t+3}) : r_t = r_{t+1} \neq a_t, r_{t+2} \neq r_{t+1}\Big\},\\
\mathcal{D}_2(\boldsymbol{m}) &= \Big\{(r_t, r_{t+1}, r_{t+2}, r_{t+3}) : r_{t+1} = r_{t+2} \neq a_t\Big\},\\
\mathcal{D}_3(\boldsymbol{m}) &= \Big\{(r_t, r_{t+1}, r_{t+2}, r_{t+3}) : r_{t+2} = r_{t+3} \neq a_t, r_{t+1} = a_t\Big\},
\end{aligned}$$

denote the three error conditions of the syndrome. Note that the marker function should satisfies the rule: $m \notin \{a, b, c\}$, if we insert m between a and b, c. So that $\mathcal{D}_1(\boldsymbol{m})$, $\mathcal{D}_2(\boldsymbol{m})$ and $\mathcal{D}_3(\boldsymbol{m})$ are pairwise disjoint sets, and represent the conditions of the marker positions when the syndrome occurred a single deletion, norm/substitution and insertion error separately and no error in the marker. If the received sequence is not in any above sets, we can see that the marker code occurred one edit error. In this situation, the separator of the next block will help the decoder to relocate the message and keep synchronization. We will cut off the sequence after r_t roughly and then decode the next block. Therefore, when one edit error occurred, the received can re-synchronize itself after decoding the current block. □

Now, we can prove theorem 1. According to lemma 3 and lemma 4, we can see that every first block is able to correct a single edit error whether the error occurred in the marker or the residue segments. And according to lemma 4 and lemma 5, after decoding the first block, we can re-synchronize the received sequence. Hence the beginning of the next block is clear, so that the blocks can be recovered in turn. However, if there are more than one error in the block, the decoder will be unable to correct the errors, and the framing of codewords will be lost, but this will be detected by the next marker and corrected accordingly.

V. Decoding of DNA Levenshtein-Marker codes

In this section, we describe a linear decoding algorithm of our *DnA-LM* code. Let $\boldsymbol{r}$ be a 4-ary sequence on Σ translated from the DNA sequencing reads according to Equation (2). We decode the *DNA-sLM* blocks iteratively from the left end of $\boldsymbol{r}$. The decoding algorithm for the *DNA-sLM* code consists of three parts: the error type and position detection, error correction, and re-synchronization, as shown schematically in Fig. 5.

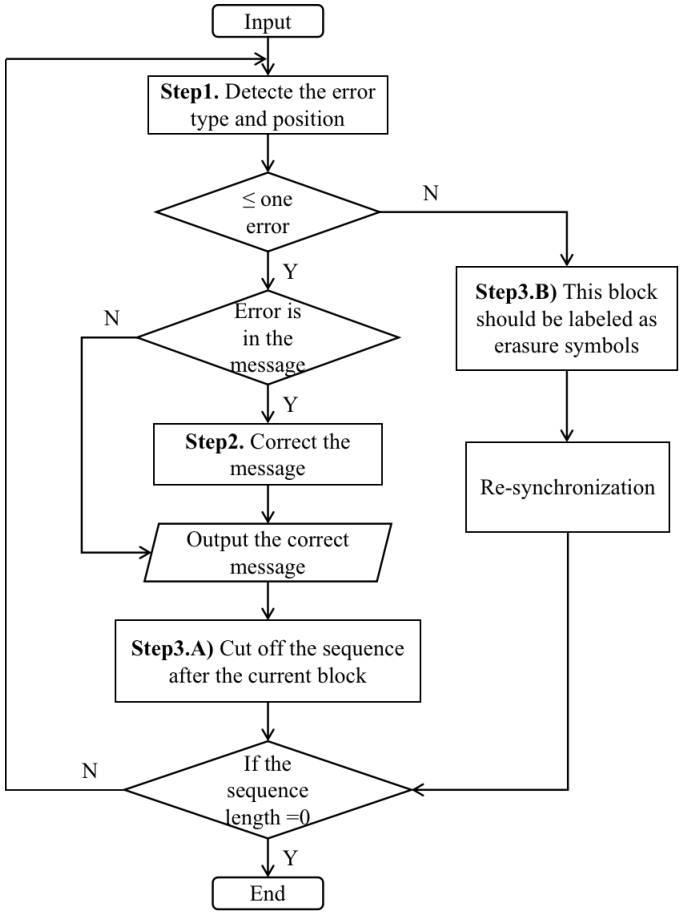


Fig. 5. The flowchart for decoding.

Step 1: Find the error type of the message. First check the first two symbols of $\boldsymbol{r}$. If $r_1 = r_2$, the marker segment is correct. We check which set S_2, S_3 the next $(k + t + 5)$ symbols belong to. Then we figure out the error type of the message following Algorithm 1. If $r_1 \neq r_2$, the error is in the marker code. The decoder will detect whether the received sequence can be matched to any of the sets in table I and locate the message position, as shown in Algorithm 2. If the received sequence can not be matched to any sets described in section III, it indicates that more than one error has occurred in the leftmost block of the sequence. Thus decoding failure happens for the current block, and the corresponding position will be labeled as erasure $\boldsymbol{e}$ in the outer code.

Algorithm 1 The Decoding Algorithm with Correct Marker

Input: a received sequence $\boldsymbol{r} = (r_1, r_2, \ldots)$, with $r_1 = r_2$.

Output: the the error type of the message segment $T \in$ {'ins','del','sub','correct','excess'}, the received message segment $\bar{\boldsymbol{x}}$.

if $r_{k+4} = r_{k+5} = r_{k+6}$ **then**
    no indel error occurred before the separator. Set $\boldsymbol{r}_k = (r_3, r_4, \ldots, r_{k+2})$, $\boldsymbol{a} = (r_{k+7}, r_{k+8}, \ldots, r_{k+6+t})$, $\overline{c} = Ck(\boldsymbol{r}_k)$, $\overline{\boldsymbol{a}} = IBSyn(\boldsymbol{r}_k)$.
    if $\overline{c} = r_{k+3}$ **then**
        if $\overline{\boldsymbol{a}} = \boldsymbol{a}$ **then**
            no error occurred in the first block. T = 'correct', $\bar{\boldsymbol{x}} = (r_3, r_4, \ldots, r_{k+2})$.
        else
            the syndrome occurred one error. T = 'correct', $\bar{\boldsymbol{x}} = (r_3, r_4, \ldots, r_{k+2})$.
    else
        if $\overline{\boldsymbol{a}} = \boldsymbol{a}$ **then**
            the check occurred one error. T = 'correct', $\bar{\boldsymbol{x}} = (r_3, r_4, \ldots, r_{k+2})$.
        else
            the message occurred one substitution. T = 'sub', $\bar{\boldsymbol{x}} = (r_3, r_4, \ldots, r_{k+2})$.
else if $r_{k+5} = r_{k+6} = r_{k+7} \neq r_{k+4}$ **then**
    one insertion error occurred before the separator. Set $\boldsymbol{r}_k = (r_3, r_4, \ldots, r_{k+2})$, $\boldsymbol{a} = (r_{k+8}, r_{k+9}, \ldots, r_{k+7+t})$, $\overline{\boldsymbol{a}} = IBSyn(\boldsymbol{r}_k)$.
    if $\overline{\boldsymbol{a}} = \boldsymbol{a}$ **then**
        the check occurred one insertion. T = 'correct', $\bar{\boldsymbol{x}} = (r_3, r_4, \ldots, r_{k+2})$.
    else
        the message occurred one insertion. T = 'ins', $\bar{\boldsymbol{x}} = (r_3, r_4, \ldots, r_{k+2}, r_{k+3})$.
else if $r_{k+3} = r_{k+4} = r_{k+5} \neq r_{k+6}$ **then**
    one deletion error occurred before the separator. $\boldsymbol{r}_k = (r_3, r_4, \ldots, r_{k+2})$, $\boldsymbol{a} = (r_{k+6}, r_{k+7}, \ldots, r_{k+5+t})$, $\overline{\boldsymbol{a}} = IBSyn(\boldsymbol{r}_k)$.
    if $\overline{\boldsymbol{a}} = \boldsymbol{a}$ **then**
        the check occurred one deletion. T = 'correct', $\bar{\boldsymbol{x}} = (r_3, r_4, \ldots, r_{k+2})$.
    else
        the message occurred one deletion. T = 'del', $\bar{\boldsymbol{x}} = (r_3, r_4, \ldots, r_{k+1})$.
else
    the separator is incorrect. Set $\boldsymbol{r}_k = (r_3, r_4, \ldots, r_{k+2})$, $\{\boldsymbol{a}\} = \{(r_{k+7+i}, r_{k+8+i}, \ldots, r_{k+6+t+i})\}_{i=-1,0,+1}$, $\overline{c} = Ck(\boldsymbol{r}_k)$, $\overline{\boldsymbol{a}} = IBSyn(\boldsymbol{r}_k)$.
    if $\overline{c} = r_{k+3}$ and $\overline{a} \in \{\boldsymbol{a}\}$ **then**
        the message is correct. T = 'correct', $\bar{\boldsymbol{x}} = (r_3, r_4, \ldots, r_{k+2})$.
    else
        more than one error in this block. T = 'excess', $\bar{\boldsymbol{x}} = \boldsymbol{e}$.
return T and $\bar{\boldsymbol{x}}$.

Algorithm 2 The Decoding Algorithm with Incorrect Marker

Input: a long sequence $\boldsymbol{r} = (r_1, r_2, \ldots)$, and $r_1 \neq r_2$

Output: the message error type $T \in$ {'correct','excess'}, the received message $\bar{\boldsymbol{x}}$..

if $r_{k+4} = r_{k+5} = r_{k+6} \neq r_{k+3}$, $r_{k+3} = Ck(r_3, r_4, \ldots, r_{k+2})$ and $(r_{k+7}, r_{k+8}, \ldots, r_{k+6+t}) = IBSyn(r_3, r_4, \ldots, r_{k+2})$ **then**
    the marker occurred one substitution, and the message is $\bar{\boldsymbol{x}} = (r_3, r_4, \ldots, r_{k+2})$. T = 'correct'.
else if $r_{k+3} = r_{k+4} = r_{k+5} \neq r_{k+2}$, $r_{k+2} = Ck(r_2, r_3, \ldots, r_{k+1})$ and $(r_{k+6}, r_{k+7}, \ldots, r_{k+5+t}) = IBSyn(r_2, r_3, \ldots, r_{k+1})$ **then**
    the marker occurred one deletion, and the message is $\bar{\boldsymbol{x}} = (r_2, r_3, \ldots, r_{k+1})$. T = 'correct'.
else if $r_{k+5} = r_{k+6} = r_{k+7} \neq r_{k+4}$, $r_{k+4} = Ck(r_4, r_5, \ldots, r_{k+3})$ and $(r_{k+8}, r_{k+9}, \ldots, r_{k+7+t}) = IBSyn(r_4, r_5, \ldots, r_{k+3})$ **then**
    the marker occurred one insertion, and the message is $\bar{\boldsymbol{x}} = (r_4, r_5, \ldots, r_{k+3})$. T = 'correct'.
else
    more than one error occurred in this block. $\bar{\boldsymbol{x}} = \boldsymbol{e}$, T = 'excess'.
return T and $\bar{\boldsymbol{x}}$

Step 2: Error correction. If there is no error in the message, T = 'correct', the decoder will return the correct message $\boldsymbol{x} = \bar{\boldsymbol{x}}$. If an error occurred in the message, $T \in$ {'ins','del','sub'}, the decoder will use the syndrome to recover the received message sequence $\bar{\boldsymbol{x}}$ through the Levenshtein decoder on the de-interleaved binary sequences. If more than one error occurred in the block, T = 'excess', set $\boldsymbol{x} = \bar{\boldsymbol{x}} = \boldsymbol{e}$.

Step 3: Re-synchronization.

A) If the current block occurred just one edit error or less, the decoder can synchronize the next block. According to lemma 5, if the received can be matched to sets $\mathcal{D}_1(\boldsymbol{m})$, $\mathcal{D}_2(\boldsymbol{m})$ or $\mathcal{D}_3(\boldsymbol{m})$, we can figure out the end of the current block. Suppose that the last symbol of the current block is r_p, the decoder will cut off the sequence after r_p. on the other hand, if we cannot find the block boundary, directly cut off the sequence after r_{k+t+6};

B) If synchronization were not intact when more than one error occurred, the decoder will scan the received sequentially until finding a subsequence which is the same as (a,m,m,b,c), here $m = Mk(a, b, c)$. And re-synchronize successfully by truncating sequence before the marker symbol m.

Step 4: Repeat steps 1-3 until the received sequence length is zero.

VI. Code Capability

A. Algorithm complexity

The encoding algorithm consists of four main components: calculating the syndrome, the check, the separator and the marker. These calculations can be accomplished in linear time. Hence the encoding of one block can be performed in $O(k)$ time. The decoding algorithm of one block can be computed

in $O(k)$ time as well. Here k is the message block length. And the decoding of sequence proceeds sequentially through blocks. Therefore, for the whole K length message string, the encoder and decoder can be performed in time $O(K)$.

B. Code rate

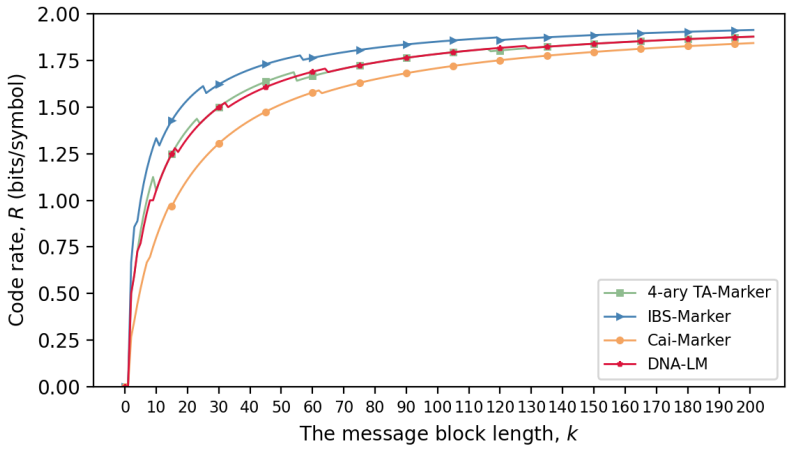
Our code string is concatenated by a series of *DNa-sLM* codeword blocks. We save the two markers in the first block to reduce redundancy without affecting the error correcting ability of the code. Let N and n denote the lengths of the whole *DNA-LM* code string and one *DNA-sLM* codeword block with K and k message symbols respectively. Let l be the number of codeword blocks in one codeword string. Note that the input and output of the code are both quaternary strings. We have,

$$\begin{aligned} K =& kl; \\ n =& k + \lceil \log k \rceil + 7; \\ N =& n - 2 + (l-1)n = l(k + \lceil \log_2 k \rceil) + 7l - 2. \end{aligned}$$

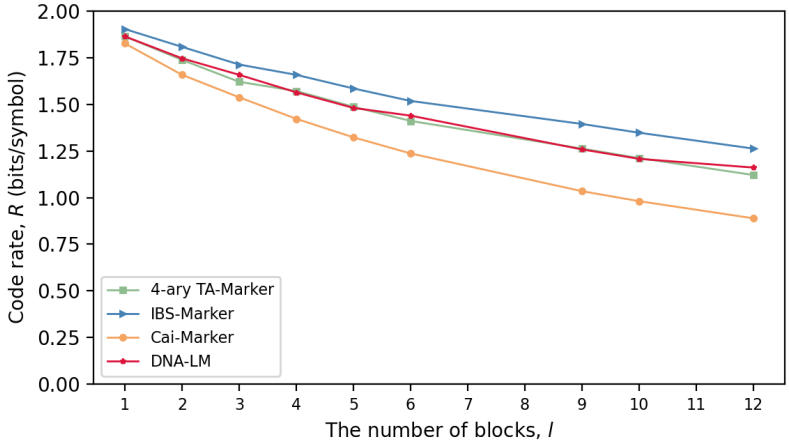
Thus, the code rate of the *DNA-LM* code which has l-blocks with k-length is

$$R = \frac{2lk}{l(k + \lceil \log_2 k \rceil) + 7l - 2} \quad \frac{\text{bits}}{\text{symbol}}. \tag{13}$$

We compare the code rate of our *DNA-LM* code with the *4-ary TA-Marker* code, the *IBS-Marker* code, *Cai-Marker* code introduced in section III with varied message block length (Fig. 6(a)). We can see that the *IBS-Marker* code achieved the highest code rate when the message block length is below 200. Our *DNA-LM* code obtained a similar code rate as the *4-ary TA-Marker* code, which is a bit lower than the *IBS-Marker* codes. The difference between the code rate of the *DNA-LM* and the *IBS-Marker* code gets smaller as the message block length increases. *Cai-Marker* code has the lowest code rate when the message block length is below 200. This is because *Cai-Marker* code requires some constant redundant symbols, although its code rate is order optimal asymptotically. We also calculated the code rate when the block number varies with the total message length K fixed (Fig. 6(b)). As expected, we observed a lower code rate when we divide the message into more blocks. Overall, we could achieve a code rate of at least 1.25 bits/symbol using no more than six blocks when encoding a quaternary message of 180 symbols. We will see that there is a trade-off between the code rate and the error rate when selecting the number of blocks l.



(a) Varied message block lengths



(b) Varied number of blocks ($K = 180$)

Fig. 6. The code rates of four concatenated codes described in this paper. Fig. (a) shows the code rate versus the quaternary message block length (in nucleotides). Here we plot the code rate of a single block containing one marker and one generalized Levenshtein structure. Fig. (b) shows the code rate versus the number of blocks with fixed message sequence length $K = 180$.

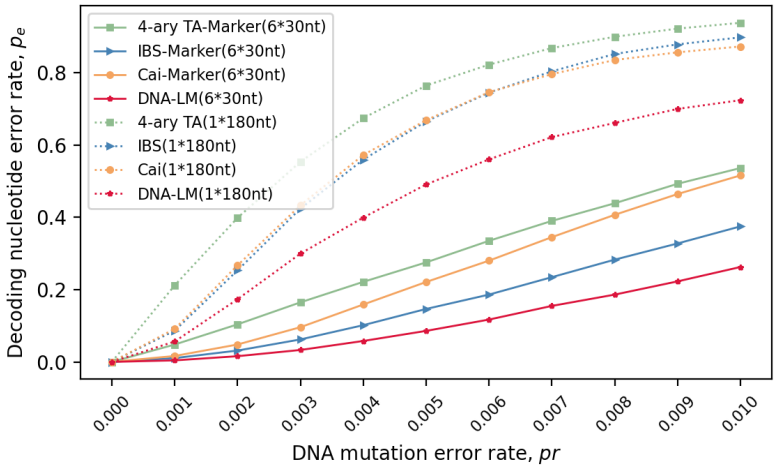
C. Success probability of decoding

Let p_i, p_d and p_s denote the rates of insertion, deletion and substitution error, and $p_c = 1 - p_i - p_d - p_s$ denote normal transmission rate. Assume that positions on the DNA string are independent to each other and that the blocks are independent to each other. The probability of having no more than one edit error in each block is

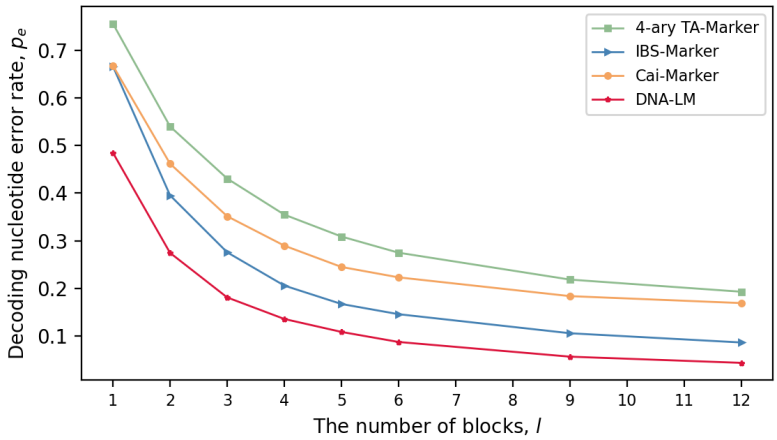
$$P_{susblock} = p_c^{n-1}\big(n - (n-1)p_c\big). \tag{14}$$

Our codeword string is the series connection of l such blocks, so we can easily calculate the probability of having no more than one edit error in each block of the whole codeword string. $P_{success} = P_{susblock}^{l}$ is the theoretical decoding success probability for our *DNA-LM* code. But in some rare situations, the decoder may not recognize the actual position of the insertion event. For example, if the previous block's last two

symbols are the same and the syndrome segment has one special insertion which causes the decoder to recognize these two equal symbols as the marker. Therefore, the success probability of decoding the received string is a little deviated from the theoretical value.



(a) Different DNA mutation error rates ($K = 180$, $k = 30$, $l = 6$)

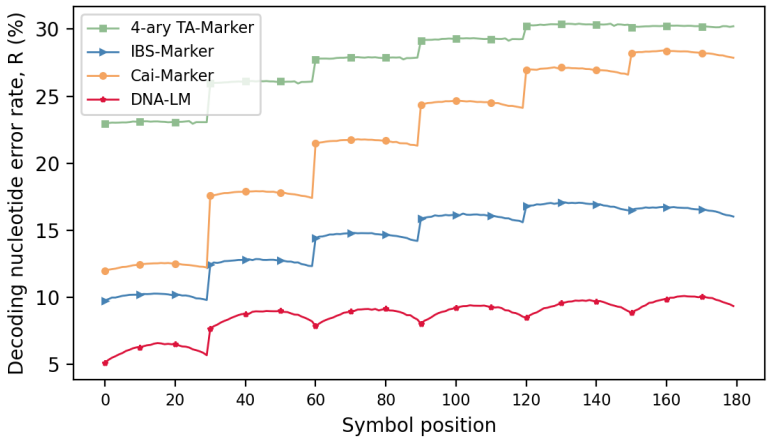


(b) Different block numbers ($K = 180$, $pr = 0.005$)

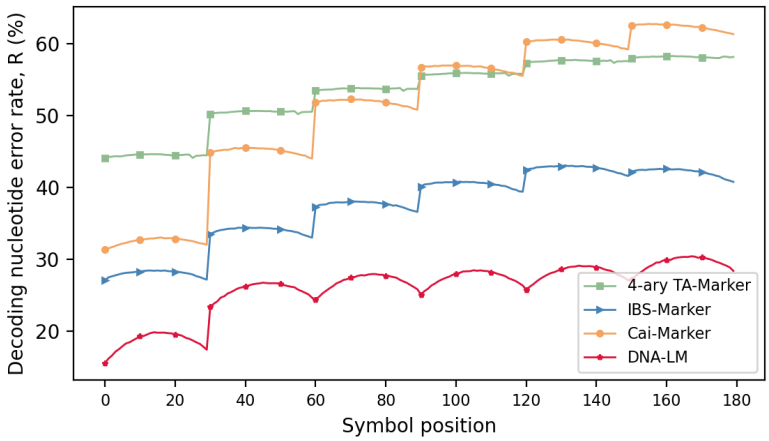
Fig. 7. Performance of the decoding nucleotide error rate of four codes described before. Figure (a) shows the nucleotide error rate as the function of DNA mutation error probability, where $p_i = p_s = p_d = pr$ increase from 0 to 0.01. Figure (b) shows the nucleotide error rate as the function of block numbers with the DNA mutation error probability $p_i = p_s = p_d = pr = 0.005$ and message length $K = 180$. The simulation times $T = 10000$.

We simulated errors on DNA strings and calculated the decoding nucleotide error rate for our *DNA-LM* code and three single-error-correction codes combined with markers (Figure 7(a)). We set the error probabilities $p_i = p_d = p_s = pr$. We define the nucleotide error rate in decoding as the ratio of string positions with the wrong nucleotide symbol. We could see that when the hyperparameters (k, l, pr) are the same, our *DNA-LM* code achieved the lowest nucleotide error rate among the four codes. The *4-ary TA-Marker* code has the highest decoding nucleotide error rate, which is expected because this code can only correct indels but not substitutions. The *IBS-Marker* code and *Cai-Marker* code achieved intermediate

nucleotide error rates. Their error rates are similar because both could correct one edit error in the codeword block but not in the markers. We also compared the decoded nucleotide error rate when the message sequence is encoded in just one block (dotted lines) versus when the message is segmented into multiple blocks (solid lines). As expected, the error rate is lower when the number of blocks is more. Figure 7(b) further illustrates this relationship between the number of blocks and the nucleotide error rate. The nucleotide error rate drops as the number of blocks increases.



(a) $pr = 0.005$



(b) $pr = 0.01$

Fig. 8. The decoding nucleotide error rates at different positions in the message. The independent variables are the positions of message symbols, and the dependent variables are percentage of the decoding error times divided by test times. The DNA mutation error probability $p_i = p_s = p_d = pr = 0.005$ in Fig.(a), $p_i = p_s = p_d = pr = 0.01$ in Fig.(b). The number of simulations $N = 100000$.

Figure 8(b) and 8(a) shows the decoded nucleotide error rate of different positions in the message sequence, when $pr = 0.005$, $pr = 0.01$ and the message is six 30-length blocks. We can see that there is a small increase in the error rate between two blocks, it is because that when the previous block occurred more than one error, the next block frame will be corrupted, which means that errors will be inherited. According to the simulation test results, compared with the other three codes, the curve of the *DNA-LM*

code is much smoother, which represents the impact of error succession on this code is minimum. Hence the remainder error-correcting procedure which includes the remainder error and the erasure recovered by outer code will be easier to process. Besides, Note that every block trend curve, especially the *DNA-LM* code, can be fitted as the normal distribution. It is the result of the systemic and splicing code structure and the distribution of indel error.

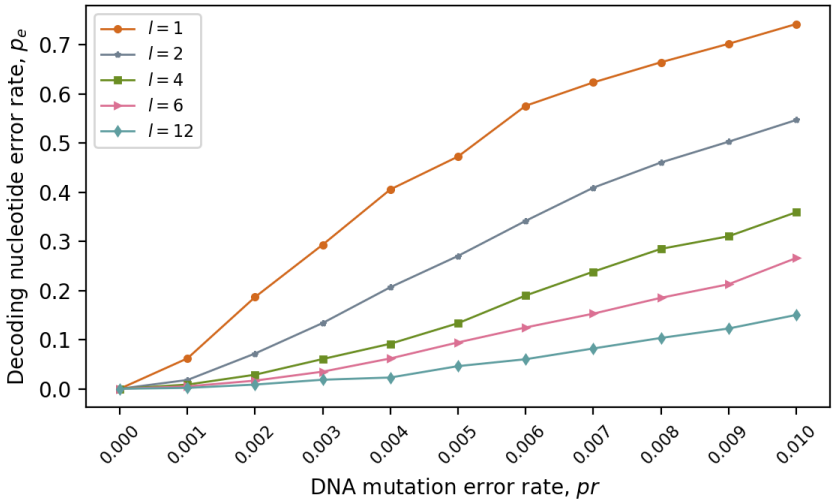


Fig. 9. The decoding nucleotide error rate of the *DNA-LM* code with different numbers of blocks l. The message length $K = 180$ and DNA mutation error rate $p_i = p_d = p_s = pr$. The number of simulation $N = 10000$.

Finally, we plot the decoding nucleotide error rate versus the DNA mutation error rate for our DNA-LM code with varied number of blocks (Fig. 9). The decoding error rate increases with the mutation error rate and decreases with the number of blocks. Recall that the code rate decreases with the number of blocks, there is a trade-off between the code rate and the nucleotide error rate. Given the total message length K, we could choose an appropriate number of blocks by considering the expected DNA mutation error rate, the minimum required code rate and the desired decoding error rate.

VII. Conclusion

In this work, we designed a code capable of correcting multiple edit errors for DNA-based storage systems. Our code, called *DNA-LM*, is concatenated by a series of single edit correcting code blocks called *DNA-sLM* codes. Our *DNA-sLM* codes are distinctive in that it does not need the received sequence length as the prerequisite information for decoding. Instead, it could identify the location of the endpoint of the codeword when there is no more than one edit error in the received sequence. This property enables

us to decode the whole DNA strand with multiple blocks iteratively from its leftmost block. Besides, our *DNA-LM* code could be encoded and decoded efficiently in linear time. Compared to other edit error correction designs in DNA storage systems, our *DNA-LM* code achieved a much lower decoding error rate with a high code rate of 1.44 bits/nucleotide when encoding a quaternary message of 180 symbols to a 250-nt DNA strand with six blocks. Our *DNA-LM* code provides a practical multiple edit error correction schema of DNA strands, which could serve as the inner code for the DNA-based storage systems and cooperate with the outer codes to achieve a desired overall information retrieval error rate.

ACKNOWLEDGMENT

We would like to thank our colleagues at the Center for Applied Mathematics, Tianjin University: Dr. Alan J.X. Guo, Dr. Huaming Wu, and Guanjin Qu who helped us a lot. This work is supported by the National Key R&D Program of China (grant No. 2020YFA0712102) and the National Natural Science Foundation of China (grant no. 12001401).

REFERENCES

[1] G. Church, Y. Gao, and S. Kosuri, "Next-generation digital information storage in dna," *Science (New York, N.Y.)*, vol. 337, p. 1628, 08 2012.

[2] N. Goldman, P. Bertone, S. Chen, C. Dessimoz, E. Leproust, B. Sipos, and E. Birney, "Towards practical, high-capacity, low-maintenance information storage in synthesized dna," *Nature*, vol. 494, 01 2013.

[3] R. N. Grass, R. Heckel, M. Puddu, D. Paunescu, and W. J. Stark, "Robust chemical preservation of digital information on dna in silica with error-correcting codes," *Angew Chem Int Ed Engl*, vol. 54, no. 8, pp. 2552–2555, 2015.

[4] Y. Erlich and D. Zielinski, "Dna fountain enables a robust and efficient storage architecture," *Science*, vol. 355, no. 6328, pp. págs. 950–954, 2017.

[5] L. Organick, S. D. Ang, Y. J. Chen, R. Lopez, S. Yekhanin, K. Makarychev, M. Z. Racz, G. Kamath, P. Gopalan, and B. Nguyen, "Random access in large-scale dna data storage," *Nature Biotechnology*, vol. 36, no. 3, 2018.

[6] I. Shomorony and R. Heckel, "Dna-based storage: Models and fundamental limits," *IEEE Transactions on Information Theory*, vol. PP, p. 1, 2 2021.

[7] S. Shin, R. Heckel, and I. Shomorony, "Capacity of the erasure shuffling channel," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020.

[8] T. Marie-Ka, B. C. Fidel, G. Nicolas, and N. Benoit, "Illumina library preparation for sequencing the gc-rich fraction of heterogeneous genomic dna," *Genome Biology Evolution*, no. 2, pp. 616–622, 2018.

[9] R. Heckel, G. Mikutis, and R. N. Grass, "A characterization of the dna data storage channel," *Scientific Reports*, 2018.

[10] H. Mercier, V. K. Bhargava, and V. Tarokh, "A survey of error-correcting codes for channels with symbol synchronization errors," *IEEE Communications Surveys and Tutorials*, vol. 12, pp. 87–96, 2010.

[11] M. Mitzenmacher, "A survey of results for deletion channels and related synchronization channels," *Probability Surveys*, vol. 6, pp. 1–33, 2009.

[12] S. B. Wicker and V. K. Bhargava, *An Introduction to ReedSolomon Codes*. An Introduction to Reed-Solomon Codes, 2009.

[13] S. Chandak, J. Neu, K. Tatwawadi, J. Mardia, and H. Ji, "Overcoming high nanopore basecaller error rates for dna storage via basecaller-decoder integration and convolutional codes," 2019.

[14] W. Press, J. Hawkins, S. Jones, J. Schaub, and I. Finkelstein, "Hedges error-correcting code for dna storage corrects indels and allows sequence constraints," *Proceedings of the National Academy of Sciences*, vol. 117, p. 202004821, 7 2020.

[15] J. L. Fan, "Array codes as low-density parity-check codes," 2000.

[16] R. G. Gallager, *Low-density parity-check codes /.* Channel Coding Techniques for Wireless Communications, 2015.

[17] J. Chen, M. Mitzenmacher, C. Ng, and N. Varnica, "Concatenated codes for deletion channels," in *IEEE International Symposium on Information Theory, 2003. Proceedings.*, 2003.

[18] M. Luby, "Lt-codes," in *Proceedings of the ACM Symposium on Foundations of Computer Science (FOCS), 2002. 08*, 2002.

[19] A. Shokrollahi, "Raptor codes," *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2551–2567, 2006.

[20] K. Cai, Y. M. Chee, R. Gabrys, H. M. Kiah, and T. T. Nguyen, "Correcting a single indel/edit for dna-based data storage: Linear-time encoders and order-optimality," *IEEE Transactions on Information Theory*, vol. 67, pp. 3438–3451, 6 2021.

[21] N. Sloane, "On single-deletion-correcting codes," *Mathematics*, pp. 273–291, 2002.

[22] V. Levenshtein, "Binary codes capable of correcting insertions and reversals," *Sov. Phys. Dokl.*, vol. 10, 1 1966.

[23] A. Helberg, "Coding for the correction of synchronization errors," 6 1993.

[24] G. Tenengolts, "Nonbinary codes, correcting single deletion or insertion (corresp.)," *Information Theory, IEEE Transactions on*, vol. 30, pp. 766–769, 10 1984.

[25] M. Abroshan, R. Venkataramanan, and A. G. I. Fabregas, "Efficient systematic encoding of non-binary vt codes," vol. 2018-June. Institute of Electrical and Electronics Engineers Inc., 8 2018, pp. 91–95.

[26] T. A. Le and H. D. Nguyen, "New multiple insertion/deletion correcting codes for non-binary alphabets," *IEEE Transactions on Information Theory*, vol. 62, pp. 2682–2693, 5 2016.

[27] D. C. H. Tan, "Vt_codes," https://github.com/shubhamchandak94/VT_codes/.

[28] D. Bar-Lev, T. Etzion, and E. Yaakobi, "On the size of levenshtein balls," 03 2021.

[29] V. Guruswami and J. Hstad, "Explicit two-deletion codes with redundancy matching the existential bound," 2020.

[30] S. Jin, R. Gabrys, and J. Bruck, "Optimal systematic t-deletion correcting codes," in *2020 IEEE International Symposium on Information Theory (ISIT)*, 2020.

[31] D. C. H. Tan, "Single edit correcting code," https://github.com/dtch1997/single-edit-correcting-code.git.

[32] F. F. S. Jr., "Bit loss and gain correction code," *IRE Trans.*, vol. IT-8, pp. 35–38, 1962. [Online]. Available: https://doi.org/10.1109/tit.1962.1057684

[33] R. R. Varšamov and G. M. T. c, "A code which corrects single asymmetric errors," *Avtomat. i Telemeh.*, vol. 26, pp. 288–292, 1965.

[34] K. Abdel-Ghaffar and H. Ferreira, "Systematic encoding of the varshamov-tenengol'ts codes and the constantin-rao codes," *Information Theory, IEEE Transactions on*, vol. 44, pp. 340 – 345, 02 1998.

[35] K. Saowapa, H. Kaneko, and E. Fujiwara, *Systematic deletion/insertion error correcting codes with random error correction capability*, 12 1999.

[36] H. Ferreira, W. Clarke, A. Helberg, K. Abdel-Ghaffar, and J. Vinck, "Insertion/deletion correction with spectral nulls," *IEEE Transactions on Information Theory*, vol. 43, p. 2, 03 1997.