OXFORD

Briefings in Bioinformatics, 2023, **24(1)**, 1–11
https://doi.org/10.1093/bib/bbac484
Problem Solving Protocol

Multiple errors correction for position-limited DNA sequences with GC balance and no homopolymer for DNA-based data storage

Xiayang Li, Moxuan Chen and Huaming Wu

Corresponding author. Huaming Wu, Center for Applied Mathematics, Tianjin University, Tianjin, 300072. E-mail: whming@tju.edu.cn

Abstract

Deoxyribonucleic acid (DNA) is an attractive medium for long-term digital data storage due to its extremely high storage density, low maintenance cost and longevity. However, during the process of synthesis, amplification and sequencing of DNA sequences with homopolymers of large run-length, three different types of errors, namely, insertion, deletion and substitution errors frequently occur. Meanwhile, DNA sequences with large imbalances between GC and AT content exhibit high dropout rates and are prone to errors. These limitations severely hinder the widespread use of DNA-based data storage. In order to reduce and correct these errors in DNA storage, this paper proposes a novel coding schema called *DNA-LC*, which converts binary sequences into DNA base sequences that satisfy both the GC balance and run-length constraints. Furthermore, our coding mode is able to detect and correct multiple errors with a higher error correction capability than the other methods targeting single error correction within a single strand. The decoding algorithm has been implemented in practice. Simulation results indicate that our proposed coding scheme can offer outstanding error protection to DNA sequences. The source code is freely accessible at https://github.com/XiayangLi2301/DNA.

Keywords: DNA-based data storage, Constrained codes, Error-detecting codes, GC-content

Introduction

With the advent of the era of big data, massive data are generated and collected every day by different types of sensors and devices, and the ever-increasing data of humans have brought huge pressure on traditional storage methods, e.g. magnetic disks, optical disks and hard disks, which are no longer sufficient for long-term digital data storage [1]. Along with the maturity of DeoxyriboNucleic Acid (DNA) synthesis and sequencing technologies, DNA emerges as a promising medium for cold data (i.e. infrequently accessed data) storage. DNA strings are composed of four types of nucleotides: Adenine (A), Thymine (T), Guanine (G) and Cytosine (C), while satisfying multiple constraints for robustness. DNA shows some remarkable and overwhelming advantages, e.g. enormous information storage density, low maintenance cost, strong stability and long longevity [2]. However, there still exist many technological difficulties in DNA storage systems that need to be overcome.

One of the dominant difficulties is that synthesizing and sequencing DNA sequences are still far from perfect [3]. During the synthesis, amplification and sequencing procedures, there may occur insertion, deletion and substitution errors of nucleotides in individual DNA molecules [4]. To tackle this challenge, many prior studies have attempted to design various error correction methods based on the biochemical characteristics of DNA. Oligos with a large unbalance between GC and AT content exhibit high dropout rates and are prone to errors [5]. In other words, DNA sequences with GC content occupying exactly 50%, also known as GC-AT balance [6] (i.e. the sum of Gs and Cs is the same as the sum of As and Ts in each DNA codeword), are much more stable than those with higher or lower GC content. In addition, ROSS *et al.* [5] also reported that repetitions of the same nucleotide, i.e. a homopolymer run-length, can significantly increase the probability of errors in sequencing. Then, the common thread of later researchers is trying to construct DNA sequences with these constraints in theory.

In recent years, there have been many studies aimed at constructing DNA sequences with multiple constraints [7–12]. For instance, Song *et al.* [7] designed a novel method to translate binary sequences into DNA sequences that satisfy the constraints of GC-content and run-length. Benerjee *et al.* [9] considered reverse and reverse-complement constraints in DNA codewords, and then designed a reverse-complement DNA code. Dube *et al.* [10] constructed a code satisfying Run-Length Limitation Three (RLL-3) constraint and Knuth-Like Balancing of the GC Contents. Chee *et al.* [11] proposed a novel approach to encoding quaternary data into strands of GC-balanced codewords, which could correct at most three long tandem duplications. Park *et al.* [12] applied the greedy algorithm to iterative encoding DNA sequences with GC balance and run-length constraints. However, all of the above studies only considered multiple constraints for DNA codewords, while ignoring error correction schemes.

The error rate of DNA storage is greatly affected by the biochemical structure of DNA [12–14]. Apart from the intrinsic features of DNA molecules for data storage, it is often required that the DNA code has certain error correction abilities in the face of biochemical errors in DNA sequences [15, 16]. There have been

Xiayang Li is an undergraduate student at the School of Mathematics, Tianjin University. His research focuses on DNA data storage.
Moxuan Chen is an undergraduate student at the School of Mathematics, TIanjin University. Her research focuses on DNA data storage.
Huaming Wu is an Associate Professor at the Center for Applied Mathematics, Tianjin University. His research interests include edge computing, internet of things and DNA storage.
Received: June 27, 2022. **Revised:** October 12, 2022. **Accepted:** October 13, 2022

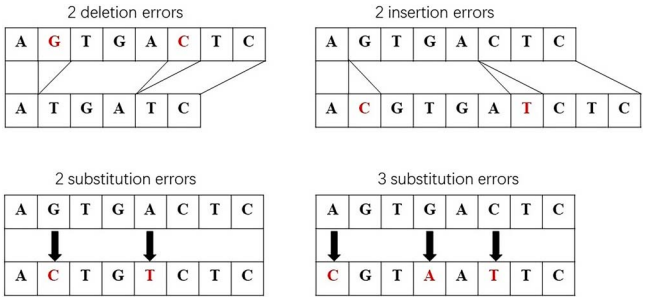

Figure 1. Multiple errors occurred within one strand.

multiple efforts that focus on error correction for DNA-based storage systems from the perspective of theory and experiments. Earlier, Levenshtein [17] first proposed a code for correcting one single synchronization error on the binary alphabet, which laid the foundation for much research later. DNA Fountain code [18, 19] was proposed for multiple errors correction/detection on the binary alphabet that enables efficient encoding of information. Chandak *et al.* [20] applied Low-Density Parity-Check codes (LDPC) [21] of a large block length to handle insertions and deletions.

By far, on the basis of the Levenshtein code, some systematic codes have been designed and practically applied to the DNA storage system [22–25]. For instance, Weber *et al.* [26] established one single-error-detecting code with GC-weight and run-length constraints for DNA storage. Xue *et al.* [27] proposed a systematic single insertion/deletion/substitution error correction code, for convenience, called *DNA-XL* in this paper, which can be used to construct GC-balanced DNA sequences. *DNA-XL* code is quite effective when the error rate is low, however, with the increase in the error rate, multiple errors are prone to occur within one single sequence. To demonstrate the motivation for our work, we list an example as shown in Figure 1, in which case the *DNA-XL* code, as well as many other DNA codes, fails to work since multiple errors occur within one strand. Recently, Nguyen *et al.* [28] presented a simple and efficient (ϵ, ℓ)-constrained encoder that can correct a single indel/edit error. Unfortunately, these existing DNA encoding methods still suffer from high complexity. What is more, the run-length constraint and GC-balanced constraint cannot be guaranteed simultaneously in the component strand.

As far as we know, no previous work has been conducted on designing DNA sequences that satisfy the above two constraints simultaneously, and in the meantime are capable of efficiently correcting multiple insertion, deletion or substitution errors. Therefore, inspired by this, we desire to design an unprecedented code with constraints enabling multiple errors correction within one strand for DNA-based data storage, where our incentive exactly lies.

In this paper, we adopted a binary encoding scheme. Moreover, in each single strand, our encoding mode called *DNA-LC* enables the detection and correction of multiple errors. To the best of our knowledge, DNA codes satisfying the two constraints for multiple errors correction within one strand have not been studied in the literature.

The main contributions of this work can be summarized as follows:

- GC-content of any DNA sequence constructed is exactly 50%, and any homopolymer is avoided in our DNA sequence.
- In each single strand, our encoding mode can detect and correct multiple errors in some situations.
- The decoding algorithm is implemented in practice. We conduct several experiments in terms of error rate for different DNA coding techniques, which show that our code achieves higher error correction capability and low time complexity.

The rest of this paper is arranged as follows: In Section 3, we introduce our *DNA-LC* code and prove its intrinsic strengths. In Section 3, we provide the proposed decoding algorithms for correcting one single error and multiple errors, which is also one of the greatest strengths of our code. In Section 3, we discuss the capabilities of *DNA-LC* code, implement performance evaluation and present one of its extensions theoretically. Section 3 summarizes the features of our *DNA-LC* code and points out the future work direction.

Encoding of our *DNA-LC* code

In this section, we first give the overall architecture of the DNA-based data storage system and review the Levenshtein code, and then we introduce our coding mode, called *DNA-LC* code, and finally prove its strengths.

System model

The general DNA-based data storage system is shown in Figure 2.

The details of the DNA-based data storage process are as follows: First, the message to be stored is converted into long binary sequences and divided into short ones after segmentation. Then, the short binary sequences are encoded and synthesized into DNA strands. Finally, the original message will be retrieved after sequencing and decoding with error correction, where short binary sequences are assembled into long binary sequences. Since multiple errors are prone to occur in synthesis and sequencing, sequence encoding and error correction are required with the inner code. In practice, in order to improve the reliability of information storage, it is necessary to detect errors, mark them as erasures, and hand them over to the outer code for correction twice. In this work, we focus on the design of the inner code.

Review of levenshtein code

Levenshtein codes are a class of binary algebraic codes that contain all binary codewords of length n satisfying as follows [17]:

$$L(n, a, U) = \left\{ x \in \{0,1\}^n : \sum_{i=1}^{n} x_i * i \equiv a \mod U \right\}. \tag{1}$$

For any integer $U \geq 2n$ and $0 \leq a \leq U-1$, it has been proven that these codes can correct a single insertion, deletion or substitution error. There have been quite numerous studies on Levenshtein codes and many excellent scholars have made great contributions in this field. Here, we only introduce what is relevant to our coding mode.

To enable the codeword to correct a single edit error, the basic idea is to insert parity check bits at the 2^i-th positions to ensure that it has the desired syndrome. What deserves our attention is that the second-to-last position is for a message bit and the last position is for a parity bit. When the length of the message bits is k, the codeword after processing has a length of n, which can be computed by

$$(\mathcal{P}) \qquad n = \min\ n', \tag{2}$$

$$s.t.: \quad k = n' - \lceil \log_2 n' \rceil - 1. \tag{3}$$

Theorem 1. For a given specific message of length k and a fixed value U with $2n \leq U \leq n + 2^{n-k-1}$, there exists at least one set of

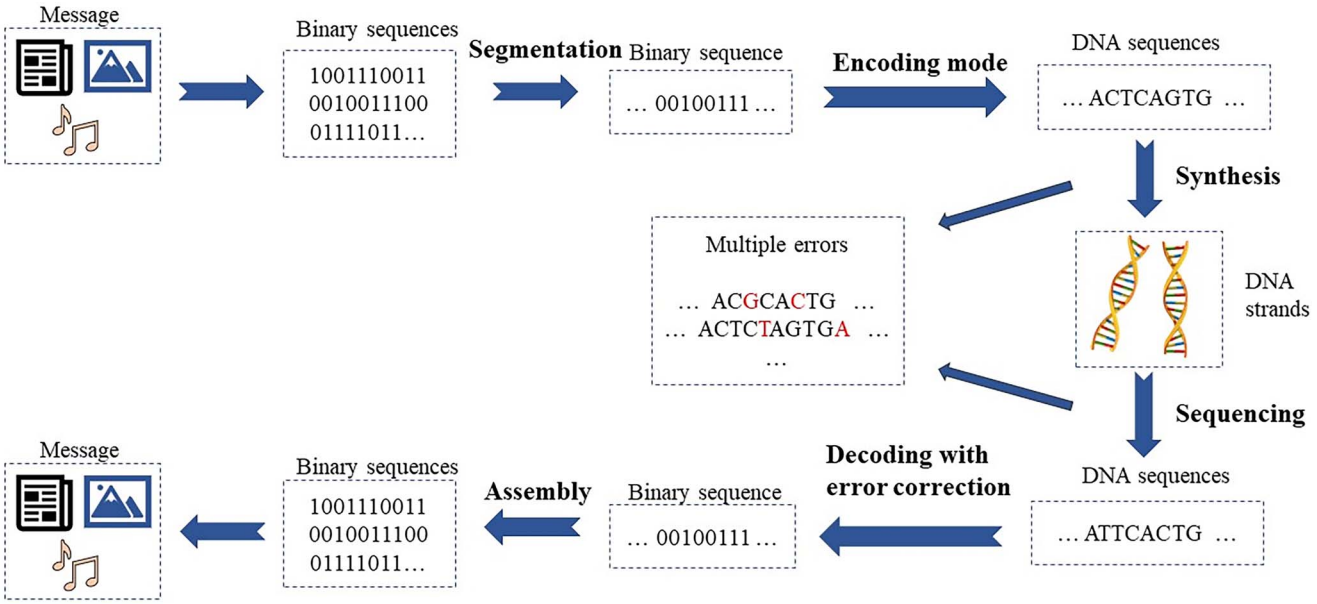

Figure 2. A typical DNA-based data storage system.

Table 1. Modulation of the bit pairs $x_{2i-1}x_{2i}$ to the nucleotide base pairs.

$x_{2i-1}x_{2i}$	00	01	10	11
base pairs	AC	AG	TC	TG

parity-check bits $\{p_1, p_2, \cdots, p_r\}$ satisfying

$$\sum_{i=1}^{n} x_i * i \equiv a \mod U. \tag{4}$$

DNA-LC code

In this subsection, we introduce the proposed coding mode that satisfies the GC content constraint, as well as the homopolymer constraint.

Let $a = 0$ and $U = 2n$ for convenience. Through the process shown above, a *k-bit* message sequence is processed into an *n-bit* codeword, which satisfies the condition as follows:

$$\sum_{i=1}^{n} x_i * i \equiv 0 \mod 2n. \tag{5}$$

Note that we require n to be an even number for our encoding mode. Theorem 2 illustrates the equivalent condition for it.

Theorem 2. We can choose an even n if and only if

$$k = 2^m + l - m - 2,\ l = 2 * 2, \cdots, 2 * 2^{m-1},\ m \geq 2.$$

Proof. Based on the Levenshtein code, we can know that $k = n - \lceil \log_2 n \rceil - 1$. Now we choose an even number $n = 2^m + l \in (2^m, 2^{m+1}]$ and m satisfying $m \geq 2$. Then we can conclude that $k = (2^m + l) - (m + 1) - 1 = 2^m + l - m - 2$. ■

The modulation strategy we adopt for the *DNA-LC* code is presented in Table 1, where bit pairs 00, 01, 10 and 11 are modulated to base pairs AC, AG, TC and TG, respectively.

Definition 1. For the four nucleotide bases, we define an A or T as an odd base, while a C or G is defined as an even base.

Definition 2. Misplacement in this paper refers to the three following cases:

1. An odd base is situated in an even position.
2. An even base is situated in an odd position.
3. An odd base is situated in the last position.

For the first two cases, we record the position of the wrong base. For the last case, we record the position after the wrong base. In our work, we need to detect and record all the misplacement positions. On the basis of these definitions, we can naturally obtain the following Lemmas.

Lemma 1. If no error has occurred, misplacement would not happen in the *DNA-LC* code, i.e. all the odd or even bases are in the corresponding odd or even positions.

Proof. As shown in Table 1, A or Ts are always situated in the odd positions, while C or Gs are in the even positions. Hence, all the odd or even bases are in the corresponding positions. ■

Lemma 2. GC-content of any *DNA-LC* code is exactly 50%, which is also referred to as GC-balanced.

Proof. For our encoding, we have required the length of the sequence n to be even. Then, according to Lemma 1, the even bases G and C are always placed in the even positions. Hence, the sum of G and Cs in any *DNA-LC* code is $n/2$, indicating the GC-content is 50%. ■

Lemma 3. Any homopolymer is avoided in our *DNA-LC* code.

Proof. As is known to all, any two adjacent positions must be one odd and one even. As illustrated in Lemma 1, the base in an odd

Table 2. Positions of the message bits in the Levenshtein sequence. The blank cells are reserved for the parity-check bits.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
x_i			0		1	0	0		1	0	1	1	0	1	1		0	

Table 3. Levenshtein sequence consisting of both message and parity-check bits. Parity bits added are special underlined.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
x_i	$\underline{0}$	$\underline{1}$	0	$\underline{1}$	1	0	0	$\underline{0}$	1	0	1	1	0	1	1	$\underline{0}$	0	$\underline{0}$

position should be an A or T, while the base in an even position should be a C or G. As a result, the bases in any two adjacent positions can never be the same. ■

Combining all the lemmas above, we finally confirm our conclusion that our *DNA-LC* code simultaneously satisfies GC-content balance and has no homopolymer. This is also one of its greatest strengths, which has not been gained with such a small complexity in the previous work.

A simple example is listed as follows, which does help make out the whole coding process in a short time.

Example 1 Assuming the message sequence is 010010110110 and $a = 0$. Solving Problem $\mathcal{P}$ shows a codeword length of $n = 18$ bits. We arbitrarily choose $U = 2n = 36$. Then we need to add six parity-check bits to satisfy Eq. (1).

Compute the syndrome of bits by

$$\sum x_i * i = 5 + 9 + 11 + 12 + 14 + 15 = 66.$$

Therefore, the blank bits in Table 2 should have the syndrome of $2U - 66 = 6$, whose binary form is 000110. Then the result is shown in Table 3.

Employing Table 1, we then encode the Levenshtein sequence 010110001011011000 into a *DNA-LC* code AGAGTC-ACTCTGAGTC AC. Conspicuously, this *DNA-LC* code is balanced with a GC-content of $9 = n/2$ and has no homopolymer.

Decoding of our DNA-LC code

The decoding methods discussed below are aimed at one strand, i.e. a short binary sequence. In this part, we first introduce the *DNA-LC* code's algorithm for correcting a single error, and on this basis, develop an algorithm for correcting multiple errors.

Let $r = (r_1, r_2, \cdots, r_n)$, $r_i \in \{0, 1\}$, for $i = 1, 2, \cdots, n$ denote a random binary sequence. Moreover, we denote S as the syndrome of a binary sequence, which is computed by

$$S = \sum_{i=1}^{l} r_i * i \mod 2n. \tag{6}$$

Compared with the general Levenshtein algorithms, our decoding algorithm not only includes the correction for one single error, but also contains multiple errors within one strand, which is actually our contribution lies in.

Decoding for one single error correction

On account of the situation where only one single error occurred, it is quite briefer and easier to carry out our algorithms, as shown in Algorithm 1. Based on the hypothesis that only one or no error occurred, the length of the received sequence, l, has three possible values n, $n-1$ and $n+1$. In order to avoid unnecessary trouble and be unified with the multiple errors correction mechanism, let us make a point. When the base in the last position denoted as j is an A or T, we still deem that a misplacement occurs and mark the misplacement position as $j + 1$. As we will see later, this is very convenient.

Algorithm 1 The single error-correction decoding of *DNA-LC*

Input: The received DNA sequence, the length of the received sequence l, and the length of the error-free sequence n.
Output: The corrected DNA sequence.

```
if l = n then
    if Any base is misplaced then
        Determine the error position
        Get two correction schemes
    else
        Execute the Levenshtein algorithm
    end if
else if l = n − 1 then
    if The base is in the last position then
        Add an even base to the last position
    else
        Add a relevant base before it
    end if
else if l = n + 1 then
    if The misplaced base in the last position then
        Delete itself
    else
        Delete itself or the one before it
    end if
else
    return decoding failed signal
end if
return the corrected sequence with S = 0
```

Lemma 4. If only one base is different between two *DNA-LC* sequences, their binary sequences differ from only one position.

Proof. As shown in Table 1, there are four cases where only one base is different, i.e. AC&AG, AC&TC, AG&TG, TC&TG. Evidently, in each case, their binary sequences only have a one-bit difference. ■

Lemma 4 ensures the general Levenshtein algorithm for single error correction is valid here. Detailed decoding of the *DNA-LC* code is presented in three cases below.

Case 1: $l = n$

Most importantly, on the basis of 'misplacement' defined in Definition 2, we can quickly determine the error position and restore the right sequence, thereby reducing the complexity of correction. Theorem 3 states the principle of this step.

Theorem 3. Assuming that only one single error has occurred, the misplacement position detected is exactly the place where a substitution error occurred. Furthermore, through calculation, we can distinguish the right one from the two correction schemes.

Proof. Obviously, the occurrence of misplacement is owed to an insertion, deletion or substitution error. Given that $l = n$, the error type must be a substitution. According to the DNA pairs set {AC, AG, TC, TG}, we have two correction schemes to try and then decode both of the *DNA-LC* sequences into binary sequences. By computing the syndrome S of them, we adopt the scheme whose syndrome S is 0. ■

Example 2. Suppose that the received DNA sequence is ACTGTT. Obviously, the last nucleotide base T is misplaced. After correction, we attain two possible sequences: ACTGTC or ACTGTG. Decode both of them into binary sequences: 001110 / 001111. Given that the length n here is 6, we arbitrarily choose $U = 12$. According to Eq. (1), we have their syndromes $S_1 = 0$ and $S_2 = 6$. So 001110 is exactly what we need.

When misplacement does not occur, we directly decode the *DNA-LC* code into a binary sequence and then employ the classical Levenshtein algorithm for correction:

- If $S = 0$, there is no error in the received sequence.
- If both $0 < S \leq n$ and $r_S = 1$ are satisfied, we turn the value of r_S into 0.
- If both $S \geq U - n$ and $r_{U-S} = 0$ are satisfied, we turn the value of r_{U-S} into 1.

Case 2: $l = n - 1$

In this case, it is judged that a deletion error has occurred. It is evident that bits before the deletion (insertion) error position are not misplaced. In view of our definition of odd and even bases, we add the relevant base before the base is misplaced. Similarly, we have two schemes for correction.

In particular, what needs to be pointed out is that if the base in the last position $n - 1$ is an A or T, as described above, we will mark the misplacement position as n. In this case, we add an even base to the n-th position, and there are two correction schemes. Decode both of them by computing the value of S to distinguish which one is correct. In order to better understand this process, we use a simple example to illustrate it.

Example 3. Supposing that the received DNA sequence is ACTGC, the length of it should be 6 and there is a deletion error occurring in it. We should add an A or T before the last position. After supplement and decoding, we get such binary sequences: 001110, 001100. Therefore, the correctly transmitted sequence is 001110.

Case 3: $l = n + 1$

In this case, it is judged that an insertion error has occurred. To quickly determine the insertion position, we now introduce Lemma 5.

Lemma 5. Denote $j(b)$ as the first misplacement position, and then we can draw a conclusion that the base inserted is $r_j(b)$ or $r_{j-1}(b)$.

Proof. Assuming that the base inserted into the *DNA-LC* code is an odd base, i.e. A or T, the insertion position may be

- one position before an odd position j', then the first misplaced base $r_j(b) = r_{j'}(b)$, so the base inserted is $r_{j-1}(b)$.
- one position before an even position j', then the first misplaced base $r_j(b) = r_{j'-1}(b)$, so the base inserted is $r_j(b)$.

The same is true for the even base. ■

On the basis of Lemma 5 illustrated above, when finding the base is misplaced, we choose to delete itself or the one before it and then decode the two sequences into binary sequences. Finally, the one satisfying S = 0 is exactly what we want.

In particular, there are two special cases where only one correct sequence is generated, which saves us the workload of calculating S. If the base in the last position $n + 1$ is an A or T, we mark the misplacement position as $n + 2$ and delete the last base directly. Similarly, if the first base is a C or G, we just need to delete it. For example, assuming that the received sequence is AGTCTCT, we can easily find that the $n + 1 = 7^{th}$ base is T. Delete it and then get the correct sequence AGTCTC without calculation.

Decoding for multiple errors correction in one strand

Systematic decoding algorithms for correcting multiple errors within one strand are also what we are interested in, which are rarely mentioned by existing literature. Fortunately, our coding mode enables multiple errors to be corrected under certain conditions, and the detailed algorithmic process is shown in Algorithm 2.

Algorithm 2 The multiple error-correction decoding of *DNA-LC*

Input: The received DNA sequence, the length of the received sequence l, the length of the error-free sequence n, and the misplacement index m.
Output: The corrected DNA sequence.

```
if m > m_0 then /*m_0 is a hyperparameter */
    return Decoding failed signal
end if
Calculate the values of parameters s and t by solving Eq. (7)
if A set of nonnegative integer solutions is found then
    Perform s steps to correct insertions and t steps to correct deletions;
    Compute the syndrome S of corrected sequences;
    if None sequence satisfies S = 0 then
        Execute the Levenshtein algorithm;
    end if
else
    return Decoding a failed signal.
end if
return the corrected sequences with S = 0.
```

The new preparations required to realize multiple errors correction are presented as follows. First, we need to find all misplacements as follows. For the definition of misplacement, refer to Definition 2. When the first misplaced base is detected, we record the position, and then delete it and move the next one to its position. Continue to detect if the new base is misplaced right here until all locations are detected. Repeat this process until all the bases are detected. The concept of the misplacement index is introduced in Definition 3 for counting.

Definition 3. Whenever a misplacement is detected in this way, the misplacement index goes up by one, which is hereinafter denoted as m in brief.

Example 4. Since this concept is indeed a bit abstract, we take a random sequence ACGCACAACTT for example. The first misplaced base is the 3-th base 'G'. Then delete it and move the next one 'C' to its position, we also find that the fourth base 'C' is misplaced. Similarly, the eighth base 'A' and the 11-th base 'T' are misplaced. What is more, the last base 'T' is an odd base, so the misplacement index goes up by one. All in all, the misplacement index of this sequence is 5, and the misplacement index list is [3, 4, 8, 11, 12].

Definition 4. Type 1 error refers to the case where an odd (even) base is substituted for an odd (even) base. And other types of substitution errors are classified as Type 2 errors.

Lemma 6. The index of misplacement caused by a Type 1 error is 0, while that of a Type 2 error is 2. What is more, the index of misplacement caused by an insertion or deletion error is 1.

Proof. Since the occurrence of a Type 1 error would not cause any misplacement, the index in this case is 0. Furthermore, according to Definition 3, one base is detected to be misplaced when a base is inserted or deleted, which indicates that the index is 1. When it comes to a Type 2 error, we can view it as a composite of an insertion error and a deletion error. So the index here is 1+1=2. ■

As mentioned above, a Type 2 error can be viewed as a composite of an insertion error and a deletion error. Hence, all possible errors can be divided into: Type 1 errors, insertion errors and deletion errors. Given that Type 1 errors can be split into an insertion error and a deletion error of the same type in adjacent positions, the mixed error of deletion and insertion we are going to talk about does not include this case, which offsets the misplacement index and causes troubles. Instead, we still view it as a Type 1 error. Thus, an insertion error or a deletion error certainly causes changes in the misplacement index without offset. Lemma 7 states the relationship among the misplacement index m and the three types of errors.

Lemma 7. Assuming that there are no consecutive insertion or deletion errors, we have $m \geq |n - l|$.

Proof. Denote s, t, w as the number of discrete insertion, deletion and Type 1 errors, respectively. Then we have

$$|n - l| = |s - t|, \quad \text{and} \quad m = s + t.$$

So we have that $m = s + t \geq |s - t| = |n - l|$. ■

Having all the preliminary work done, we can finally display our algorithm for multiple errors correction. The parameters mentioned in the following part are consistent with their meanings in the preceding section. Once the DNA sequence is received, we will correct it in the following four steps.

- **Step 1**: We first need to calculate the values of parameters s and t. We need to solve the equations as follows:

$$\begin{cases} s - t = n - l, \\ s + t = m. \end{cases} \tag{7}$$

  If no non-negative integer solution is found, an uncorrectable error has occurred. If a set of non-negative integer solutions is found, then proceed to Step 2.
- **Step 2**: Find out all the positions of misplacement. At m positions of misplacement, we perform s steps to correct insertion errors and perform t steps to correct deletion errors.
- **Step 3**: Decode all of these we got in Step 2, and then we get no more than $2 * 2^{C_m^s}$ feasible solutions in total. Computing their syndrome S, we output the sequences satisfying $S = 0$.
- **Step 4**: If in Step 3 we get no sequences finally, we regard it as a Type 1 error, then we do the single error-correction decoding of our mode. If in Step 3 we get at least a sequence satisfying $S = 0$, pass Step 4.

Assuming that the number of sequences that the output contains the correct sequence is v, then the probability of successful corrections is $1/v$.

Theorem 4. When there are $t(s)$ inconsecutive deletion (insertion) errors occurring in one strand, we can recognize all the errors correctly. Particularly, the probability of success corrections is 100% when $t(s) = 2$.

Proof. Since the situation where multiple insertion errors occurred is parallel with that of deletion errors, we just study the situation of deletion errors here. Firstly, we detect the strand to obtain the value of m. Then we work out the value of t from $t = m$ and restore the strand into possible *DNA-LC* sequences of length n. According to Lemma 4, there are at most t bits differences among their binary sequences. Denote $i_1, i_2, \cdots, i_m$ as the restored positions. Let

$$\begin{aligned} S' &= x_{i_1} * i_1 + x_{i_2} * i_2 + \cdots + x_{i_t} * i_t \\ &= [i_1, i_2, \cdots, i_t] \begin{bmatrix} x_{i_1} \\ x_{i_2} \\ \vdots \\ x_{i_t} \end{bmatrix}, \end{aligned} \tag{8}$$

where $x_{i_1}, x_{i_2}, \cdots, x_{i_t} \in \{0, 1\}$.

- If $t = 2$, then $S' = x_{i_1} * i_1 + x_{i_2} * i_2$, whose possible values are 0, i_1, i_2, $i_1 + i_2$. When $i_1 \neq i_2$, these values are different from each other. Moreover, the gap among them does not exceed U. So we can use Eq. (5) to determine the correct sequence.
- If $t > 2$, we denote $(x_{i_1}, x_{i_2}, \cdots, x_{i_t})$ as the one that enables $S = 0$. Supposing that $x_{i_{k_1}} = x_{i_{k_2}} = \cdots = x_{i_{k_p}} = 1$, $x_{i_{l_1}} = x_{i_{l_2}} = \cdots = x_{i_{l_q}} = 0$. If the equation

$$[i_{k_1}, i_{k_2}, \cdots, i_{k_p}]\boldsymbol{x} = [i_{l_1}, i_{l_2}, \cdots, i_{l_q}]\mathbf{y} + kU \ (k \in \mathbb{N})$$

  has no solution, where

$$\begin{aligned} \boldsymbol{x} &= [a_1, a_2, \cdots, a_p]^T, a_1, a_2, \cdots, a_p \in \{0, 1\}, \\ \mathbf{y} &= [b_1, b_2, \cdots, b_p]^T, b_1, b_2, \cdots, b_p \in \{0, 1\}, \end{aligned}$$

  we can correct errors and get a unique sequence. If the equation has r solutions, we get $r + 1$ sequences corrected. More interestingly, if we record the remainder of the sequence divided by $2^n U$, we only need to record this information with n bits, but we can use this information to correct at most 2^{n+1} inconsecutive deletion (insertion) errors based on the fact that the gap among all S of feasible solutions won't

exceed $2^n U$. We suppose that the practical significance of this discovery is great. ■

Theorem 5. Our algorithm can correct a Type 2 error with a successful probability of 100%.

Proof. When a Type 2 error occurred, the misplacement index m is 2. Denote i_1, $i_1 + 1$ as the misplacement positions.

- Suppose that the error in position i_1 is an insertion error, while the one in position $i_1 + 1$ is a deletion error. In this way, we carry out corresponding correction procedures and decode the sequences into binary sequences. Compared with the correct one, they can be different in positions $i_1 - 1$, i_1.
- Suppose that the error in position i_1 is a deletion error, while the one in position $i_1 + 1$ is an insertion error. In this way, we carry out corresponding correction procedures and decode the sequences into binary sequences. Compared with the correct one, they can be different in positions i_1, $i_1 + 1$.

To conclude, there are at most two bits differences between the correct one and the restored sequences. For convenience, we denote i'_1, $i'_1 + 1$ as the two positions. And

$$\Delta S_{\mathbf{max}} \leq i'_1 + i'_2 \leq U. \tag{9}$$

■

In a nutshell, our algorithm for multiple errors correction is able to correct all types of one single error. As illustrated above, the steps of correcting a Type 1 error, an insertion or a deletion error are consistent with the algorithm for one single error. As for Type 2 error, Theorem 5 has proved that it can be successfully corrected.

Performance evaluation

In this section, we will conduct extensive experimental simulations to evaluate the error performance of the proposed systematic *DNA-LC* code in comparison with previously well-known encoding schemes:

- *DNA-XL* code [27]: It represents a class of methods that can correct a single error.
- *HEDGES* code [29]: It is a well-designed convolutional code, and able to correct insertions and deletions directly in a single strand of DNA.

In addition, we also compare the time complexity of different types of error-correcting codes.

Metrics

Let P_s, P_d and P_i denote the rate of substitution, deletion and insertion error, respectively, and P_c denotes normal transmission rate. In particular, the two kinds of substitution errors mentioned above are not distinguished here.

As is known to all, if a DNA sequence with a length n is uncoded, the probability of having no edit error in a strand is calculated by

$$P_{uncoded} = (P_c)^n. \tag{10}$$

The probability of having no more than one edit error in a strand is calculated by

$$P_{single} = P_c^{n-1}(n - P_c(n-1)). \tag{11}$$

The error rate is used as a metric to evaluate the performance of the *DNA-LC* code. By default, each position has the same error probability. The longer the length, the higher the error rate.

Parameter setting

Both *DNA-LC* and *HEDGES* codes can correct multiple errors, and they all have hyperparameters that reflect a tradeoff between computational workload and decoding probability. Therefore, in this part, we introduce the parameter design adopted in this paper.

- m **is the mistake index** that reflects the number of misplacement detected in the *DNA-LC* code. The mistake index is related to the number of feasible solutions to be identified whether their syndrome is 0 or not. In practice, we bound m, which means that we limit the number of errors we can handle and if m exceeds the pre-set value, we then declare the decoding failure. The rationality we do this is as follows: (i) The *DNA-LC* algorithm is theoretically guaranteed to have no more than two error positions in a single DNA sequence, especially insertions and deletions. However, if the number of error positions in a single DNA sequence is beyond a certain level, usually five or more, as shown in Figure 5, the reliability of the *DNA-LC* algorithm will be very low. So, bounding m avoids unnecessary waste of time on problems that are not suitable for the algorithm. (ii) At the same time, from the perspective of erasure-correcting codes, such as RS code, it can correct twice as many erasures as substitutions; for the outer code, the residual erasures are less harmful than substitution. It is more efficient to only detect these indel errors, then mark them as erasures and hand them over to the outer code for recovery. (iii) The analysis of existing DNA sequencing experimental data indicated that the conventional DNA-based storage system suffers a low raw error rate, almost 1% [4, 18]. So the number of error positions is not large in a single DNA strands, especially including tens of nucleotides.
- P_{ok} **is a 'greediness' parameter** that mitigates the tendency of the heap to expand exponentially. Good values for various code rates are given on strand lengths in the range of hundreds of nucleotides in the third column of Table 4 [29], and we have tuned it for best performance (See Fig. 3) to balance computational workload and decode probability, on strand lengths in the range of tens of nucleotides.
- H_{limit} **is the pre-set size of heap in the *HEDGES* code**. If the heap is larger than H_{limits}, then declare a decode failure in the *HEDGES* code. Since our generated DNA sequence is not long, with tens of nucleotides, the default number $H_{limit} = 10^5$ is reasonable.

For comparison under the same conditions, we adopt a modification of the *HEDGES* algorithm with a coding rate of 0.5, a GC content of 40–60%, no homopolymer and without any duplication and outer code. So we select the pattern '$1, 1, \cdots$' in *HEDGES* code. What is more, the sliding window to observed local constraints

Table 4. Parameters used for the simulations

Parameter	Paraphrase	Default
m	the mistake index in *DNA-LC*	5
P_{ok}	greediness parameter in *HEDGES*	−0.07
H_{limit}	the size of heap in *HEDGES*	10^5
Pattern	the encoding pattern in *HEDGES*	$1, 1, \cdots$
The window width	a window to observe local constraints	40
Runout bytes	message zeros padded in *HEDGES*	2

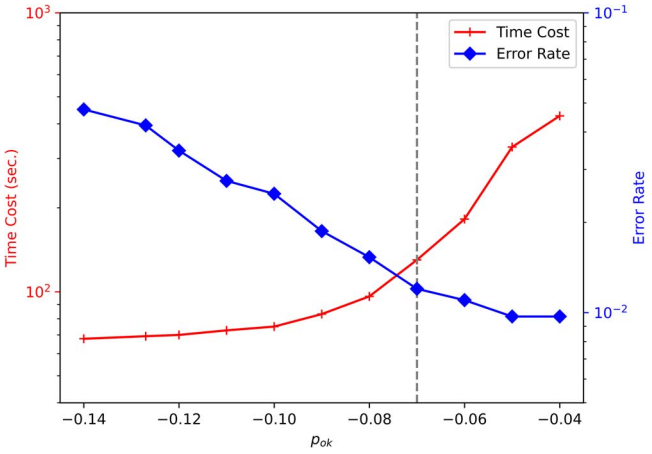

Figure 3. Change p_{ok} from -0.14 to -0.04, and observe the time cost of error correction and the variation of the error rate with p_{ok}. 33 message bits are encoded in HEDGES code.

was set as 40 in the *HEDGES* code. This is a relatively relaxed constraint. Besides, in the *HEDGES* code, the rate of decoding errors rises in the last several bytes of the message, because some incorrect chains do not have time to accumulate bad scores. To counter this, two 'runout bytes' of message zeros are padded in each strand at encoding. In this paper, we also tried one runout byte.

Impact of the length of DNA sequences

As depicted in Figure 4, we plot the decoding nucleotide error rate for different lengths of DNA sequences assuming that $P_s = P_d = P_i = 5 * 10^{-3}$. The length of DNA sequences increases from 20 to 150 nt. When a DNA sequence is 150 bases long, the error rate of *DNA-LC* is below 20%. However, the error rate of the uncoded DNA sequence is about 90%. The effect of DNA storage at this point is so poor.

Experimental verification of theorems

According to Theorem 4, when two inconsecutive deletion (insertion) errors happened, the probability of success is 100% no matter how the length of the *DNA-LC* code varies. We generated 10 000 binary sequences containing 33 message bits and encoded them into DNA sequences of 40 bases in the *DNA-LC* code. We randomly applied some inconsecutive insertions or deletions to each sequence, and found that the error correction rate was 100% when two inconsecutive insertions occurred in one single strand. As a comparison, we also generated 10 000 binary sequences of the same length, which were then encoded into *HEDGES* code. The relative results are displayed in Figure 5, which shows that our code is sensitive to deletions (similarly to insertions), especially inconsecutive, but both the *DNA-LC* code and the *HEDGES* code perform poorly when the number of errors in a single sequence is

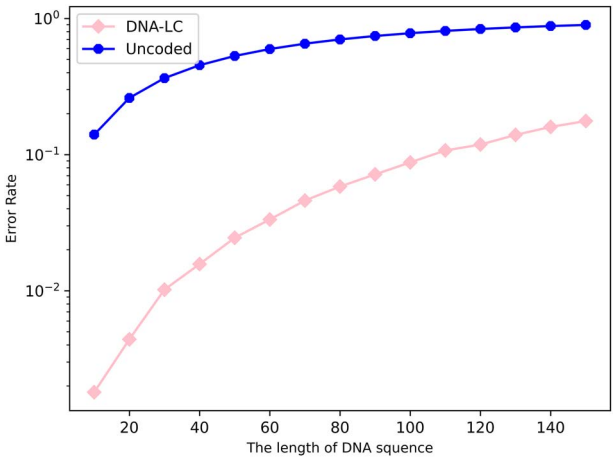

Figure 4. $P_s = P_d = P_i = 5 * 10^{-3}$, the length of DNA sequences increases from 10 nt to 150 nt.

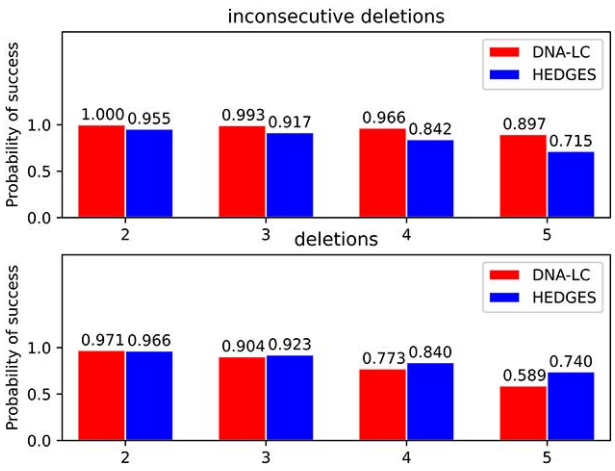

Figure 5. The error correction success rate as a function of the number of (inconsecutive) deletions.

a lot. It also reflects the rationality of our previous restriction on m. Besides, according to Figure 5 bottom, the *DNA-LC* code does not perform well while allowing consecutive types of deletions to occur.

Performance comparison

A hypothesis was made that all three types of errors would occur with the equal probability increasing from 10^{-3} to 10^{-2}, and we conducted 10 000 times simulations for each. Now 33 message bits are encoded into a DNA sequence of 25 bases in *DNA-XL* code, while they are encoded into a DNA sequence of 40 bases in our work. The results can be seen in Figure 6, showing that our proposed design can offer better error protection to DNA sequences. When compared with *DNA-XL* code that can only correct a single error with G-C balanced for DNA-based data storage, our proposed error-correcting coding scheme is capable of correcting certain multiple errors, while the coding rate is sacrificed. We also compared the error rate between the *DNA-LC* code with the *HEDGES* code. Due to the existence of runout bytes, although both the *DNA-LC* and *HEDGES* code enjoy One-Half code rates, when 33 message bits are encoded into DNA sequences, 41 bases are needed in *HEDGES* with one outrun byte (8 bits) and 49 bases are needed in *HEDGES* with two outrun bytes (16 bits), 40

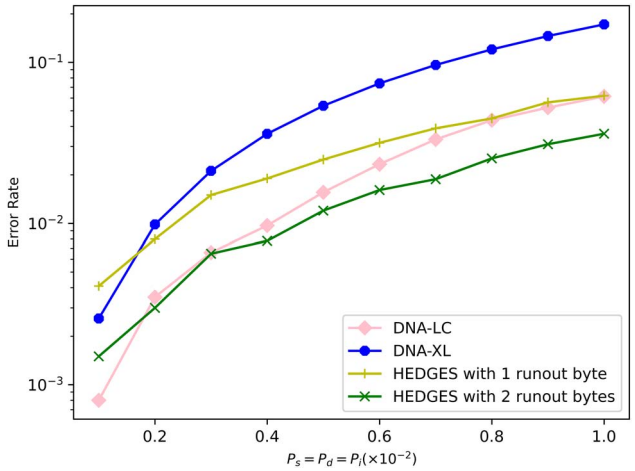

Figure 6. Performance of different codes with $P_s = P_d = P_i$ increasing from 10^{-3} to 10^{-2}.

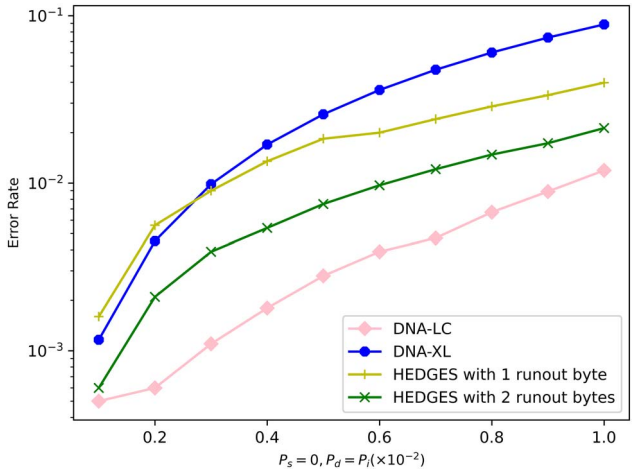

Figure 7. Performance of different DNA codes with $P_d = P_i$ increasing from 10^{-3} to 10^{-2} and $P_s = 0$.

bases are needed in the *DNA-LC* code. At the same time, the *DNA-LC* code could get better performance than the *HEDGES* code with one outrun byte. The results reflect that the need for runout bytes makes the *HEDGES* algorithm inefficient (and thus unsuitable) for an application needing short DNA strands (e.g. tens rather than hundreds of nucleotides) [29].

To demonstrate the *DNA-LC* code can better handle deletion and insertion errors in DNA-based storage systems, we also encode 33 message bits into the *DNA-XL* code, *DNA-LC* code and *HEDGES* code. To be specific, let $P_d = P_i$ increase from 10^{-3} to 10^{-2} and $P_s = 0$. Here, we made an assumption that only deletion and insertion errors occurred. Likewise, we did 10 000 times of simulations for each error rate, whose results are displayed in Figure 7. The gap is evident. When $P_d = P_i < 10^{-3}$, the error rate of *DNA-LC* is no more than 0.1%. When $P_d = P_i = 10^{-2}$, the error rate of *DNA-LC* code is no more than 2%, while that of *DNA-XL* is nearly 10%. In particular, the *DNA-LC* code could get better performance than the *HEDGES* code both with one outrun byte and two outrun bytes. In view of our error correction mechanism, we can easily conclude that our algorithm is more efficient when dealing with insertion and deletion errors.

Table 5. Comparison of edit correction coding schemes

Coding Schemes	Error correction	Decoding complexity
RS code [10]	substitutions	$O(n \log n)$
Convolutional code [29]	stochastic edit errors	$O(2^n)$
Sima *et al.* [25]	t-deletions	$O(n^{2t+1})$
Cai *et al.* [22]	single edit error	$O(n)$
Xue *et al.* (*DNA-XL*) [27]	single edit error	$O(n)$
This work (*DNA-LC*)	stochastic edit errors	$O(n)$

Time complexity comparison

Some coding schemes that perform well in errors correction may not be suitable for DNA storage due to their decoding complexity. In this respect, the *DNA-LC* code we propose enjoys low time complexity. Both encoding and decoding complexity in the *DNA-LC* code is linear. Table 5 shows the time complexity comparison among different codes.

The encoding scheme in DNA-LC code inherits the Levenshtein strategy, and then the binary sequence is converted bit-for-bit into a DNA sequence. So the conclusion that the time complexity of encoding is linear is trivial. The decoding scheme actually consists of two parts: errors detection and correction. For errors detection, the time complexity is $O(n)$ since we probe the location of the errors sequentially from the beginning to the end of the sequence. For errors correction, we need to correct the errors that occur, resulting in several probably correct solutions. Then, for each feasible solution, determine whether it is the optimal solution with $S = 0$. Obviously, the best time complexity is $O(n)$, where only two sequences need to be identified, but the worst time complexity is also related to the number of feasible solutions, which is related to the misplacement index, denoted as m. As mentioned in the previous section on parametric design, we bound m in practice, then in turn we bound the number of feasible solutions to be identified. So the worst time complexity is $O(n)$ too.

We also tested the error correction time of *DNA-LC* code and *HEDGES* code under the same platform and experimental environment. Assume that the probability of error in each DNA sequence is $p_s = p_d = p_i = 5 * 10^{-3}$. We changed the length of DNA sequence from 10 to 50 nt. We denote the time for error correction of a 10 nt long DNA sequence using the *DNA-LC* algorithm as unit time. As shown in Figure 8, it can be found that with the increase of DNA sequence length, the error correction time of *HEDGES* code changes exponentially, while the change of *DNA-LC* code is very gentle. The gap between the error correction time of the two algorithms is getting larger. The results illustrate the advantage of the lower complexity of the *DNA-LC* code, especially for DNA sequences with low error rates.

Extension of DNA-LC code

Admittedly, our *DNA-LC* code still has a shortcoming of a low storage density. Based on hachimoji DNA with eight building bases [30], we broaden the previous odd and even base subset to {A, T, P, Z} and {C, G, B, S}, respectively. The mapping is embodied in Table 6.

The design of the coding scheme satisfies the condition of Lemma 4. Therefore, it can be found that the new code inherits the strengths of the *DNA-LC* code and the algorithm for correcting one single error. If the *DNA-LC* encoding scheme is generalized to hachimoji DNA, the storage density can be improved. Since the multi-error correction algorithm is actually a single-error error correction step by step according to the error location, our proposed multi-error correction framework can also be extended

Table 6. Modulation of the bit pairs $x_{3i-2}x_{3i-1}x_{3i}$ to the nucleotide base pairs.

$x_{3i-2}x_{3i-1}x_{3i}$	000	001	010	011	100	101	110	111
base pairs	AC	AG	TC	TG	PB	PS	ZB	ZS

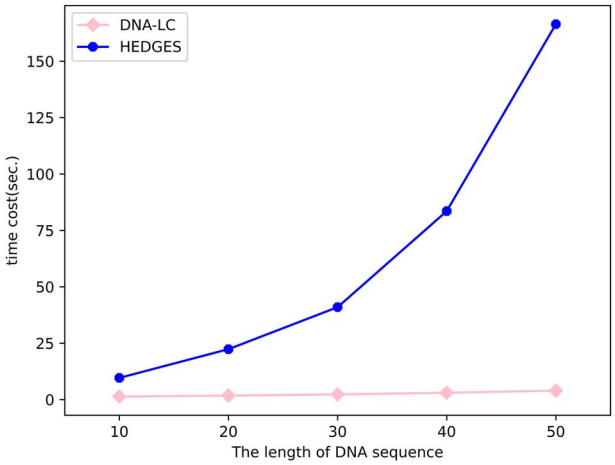

Figure 8. The consumption of decoding time varies with the length of the DNA sequence in *DNA-LC* code and *HEDGES* code.

to Hachimoji DNA indiscriminately. In the 4-base system, two nucleotides can store 2 bits of encoded information, but for the 8-base system, two nucleotides can store 3 bits. As mentioned in [30], the measured thermodynamic parameters predict the stability of hachimoji duplexes, allowing hachimoji DNA to increase the information density of natural terran DNA. However, we lack the information illustrating the condition of its stability. With the progress of life science and the improvement of DNA synthesis and sequencing technology, the use of artificial bases to store information may also be a way to expand DNA storage technology.

Conclusion

In this work, we designed a systematic code called *DNA-LC*. Constructed on the basis of the Levenshtein code, our *DNA-LC* code is excellent in that it is GC-balanced and avoids any homopolymer. It also has a relatively stable error correction ability, low time complexity and high recognition ability for insertion and deletion errors. However, the *DNA-LC* code is insensitive to consecutive errors. The proposed *DNA-LC* code is highly scalable and can be easily extended to hachimoji DNA with eight building bases to improve the storage density. For future work, we will further improve the code's ability to identify sequence reconstruction errors in DNA data storage.

Key Points

- We employed a binary encoding scheme under which the constructed DNA sequence had exactly 50% GC content and any homopolymer is avoided in the DNA sequence.
- We designed an error-correcting code for this encoding scheme. In each single strand, our encoding mode can detect and correct multiple errors in some situations. Moreover, a theoretical analysis of error correction performance is provided.
- We have implemented the decoding algorithm in practice to test the theory. We conduct several experiments in terms of error rate for different DNA coding techniques. Experiment results demonstrate that our code achieves high error correction capability and low time complexity, especially for inconsecutive insertions and deletions.

Supplementary Data

Supplementary data is available online at https://github.com/XiayangLi2301/DNA.

Author contributions statement

X. L. and H. W. conceived and designed the study. X. L. and M. C. conducted the experiments and analyzed the theoretical results. X. L., M. C., and H. W. wrote the manuscript. H. W. reviewed the manuscript.

Funding

This work is supported by the National Key Research and Development Program of China (# 2020YFA0712100) and the National Natural Science Foundation of China (# 62071327).

References

1. Yan Z, Liang C, Wu H. Upper and lower bounds on the capacity of the dna-based storage channel. *IEEE Communications Letters* 2022;1–1.
2. Qu G, Yan Z, Wu H. Clover: tree structure-based efficient DNA clustering for DNA-based data storage. *Brief Bioinform*, 2022;**23**(5):bbac336.
3. Yazdi SMHT, Kiah HM, Garcia-Ruiz E, *et al.* Dna-based storage: Trends and methods. *IEEE Transactions on Molecular, Biological and Multi-Scale. Communications* 2015;**1**(3):230–48.
4. Heckel R, Mikutis G, Grass RN. A characterization of the DNA data storage channel. *Sci Rep* 2019;**9**(1):1–12.
5. Ross MG, Russ C, Costello M, *et al.* Characterizing and measuring bias in sequence data. *Genome Biol* 2013;**14**(5):1–20.
6. Schouhamer KA, Immink, Cai K. Efficient balanced and maximum homopolymer-run restricted block codes for dna-based data storage. *IEEE Communications Letters* 2019;**23**(10):1676–9.
7. Song W, Kui Cai M, Zhang CY. Codes with run-length and gc-content constraints for dna-based data storage. *IEEE Communications Letters* 2018;**22**(10):2004–7.
8. Yixin Wang M, Noor-A-Rahim EG, Guan YL, *et al.* Construction of bio-constrained code for dna data storage. *IEEE Communications Letters* 2019;**23**(6):963–6.
9. Benerjee KG, Banerjee A. On dna codes with multiple constraints. *IEEE Communications Letters* 2021;**25**(2):365–8.
10. Dubé D, Song W, Cai K. Dna codes with run-length limitation and knuth-like balancing of the gc contents. In: *Symposium on Information Theory and its Applications (SITA), Japan*, 2019.
11. Chee YM, Chrisnata J, Kiah HM, *et al.* Efficient encoding/decoding of gc-balanced codes correcting tandem duplications. *IEEE Transactions on Information Theory* 2020;**66**(8):4892–903.

12. Park S-J, Lee Y, No J-S. Iterative coding scheme satisfying gc balance and run-length constraints for dna storage with robustness to error propagation. *Journal of Communications and Networks* 2022;**24**(3):283–91.
13. Alsaffar MM, Hasan M, McStay GP, *et al.* Digital dna lifecycle security and privacy: an overview. *Brief Bioinform* 2022; **23**(2):bbab607.
14. Hu J, Zhong Y, Shang X. A versatile and scalable single-cell data integration algorithm based on domain-adversarial and variational approximation. *Brief Bioinform* 2022;**23**(1): bbab400.
15. Ping Z, Chen S, Zhou G, *et al.* Towards practical and robust DNA-based data archiving using the yin–yang codec system. *Nat Comput Sci* apr 2022;**2**(4):234–42.
16. Zhang H, Lan Z, Zhang W, *et al.* Spider-web enables stable, repairable, and encryptible algorithms under arbitrary local biochemical constraints in dna-based storagearXiv preprint arXiv:2204.02855. 2022.
17. Levenshtein VI. Binary codes capable of correcting deletions, insertions, and reversals. In: *Soviet physics doklady*. Soviet Union, 1966;**10**(8):707–10.
18. Erlich, Y, Zielinski, D. DNA fountain enables a robust and efficient storage architecture. *Science* 2017;**355**(6328): 950–4.
19. Jeong J, Park S-J, Kim J-W, *et al.* Cooperative sequence clustering and decoding for dna storage system with fountain codes. *Bioinformatics* 2021;**37**(19):3136–43.
20. Chandak, S, Tatwawadi, K, Lau, B, *et al.* Improved read/write cost tradeoff in dna-based data storage using ldpc codes. In: *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2019, 147–56.
21. Gallager R. Low-density parity-check codes. *IRE Transactions on Information Theory* 1962;**8**(1):21–8.
22. Cai K, Chee YM, Gabrys R, *et al.* Correcting a single indel/edit for dna-based data storage: Linear-time encoders and order-optimality. *IEEE Transactions on Information Theory* 2021;**67**(6): 3438–51.
23. Bar-Lev D, Etzion T, Yaakobi E. On levenshtein balls with radius one. In: *2021 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2021, 1979–84.
24. Guruswami V, Håstad J. Explicit two-deletion codes with redundancy matching the existential bound. *IEEE Transactions on Information Theory*. 2021;**67**(10):6384–94.
25. Sima J, Gabrys R, Bruck J. Optimal systematic t-deletion correcting codes. In: *2020 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2020, 769–74.
26. Weber JH, De Groot, Van Leeuwen. On single-error-detecting codes for dna-based data storage. *IEEE Communications Letters* 2020;**25**(1):41–4.
27. Xue T, Lau FCM. Construction of gc-balanced dna with deletion/insertion/mutation error correction for dna storage system. *IEEE Access* 2020;**8**:140972–80.
28. Nguyen TT, Cai K, Schouhamer KA, *et al.* Constrained coding with error control for dna-based data storage. In: *2020 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2020, 694–9.
29. Press WH, Hawkins JA, Jones SK, *et al.* Hedges error-correcting code for dna storage corrects indels and allows sequence constraints. *Proc Natl Acad Sci* 2020;**117**(31):202004821.
30. Hoshika S, Leal NA, Kim M-J, *et al.* Hachimoji DNA and RNA: A genetic system with eight building blocks. *Science* 2019; **363**(6429):884–7.