Computational and Structural Biotechnology Journal 23 (2024) 140–147

Contents lists available at ScienceDirect

Computational and Structural Biotechnology Journal

journal homepage: www.elsevier.com/locate/csbj

Research Article

On secondary structure avoidance of codes for DNA storage

Rui Zhang [a], Huaming Wu [b,*]

[a] Chern Institute of Mathematics, Nankai University, Tianjin, 300071, China
[b] Center for Applied Mathematics, Tianjin University, Tianjin, 300072, China

ARTICLE INFO

Keywords:
DNA data storage
Secondary structure avoidance
Sequence replacement
DNA codes

ABSTRACT

A secondary structure in single-stranded DNA refers to its propensity to undergo self-folding, leading to functional inactivity and irreparable failures within DNA storage systems. Consequently, the property of secondary structure avoidance (SSA) becomes a crucial criterion in the design of single-stranded DNA sequences for DNA storage, as it prohibits the inclusion of reverse-complement subsequences that contribute to such structures. This work is specifically focused on addressing the avoidance of secondary structures in single-stranded DNA sequences. We propose a novel sequence replacement approach, which successfully resolves the SSA problem under conditions where the stem exceeds a length of $2\log_2 n + 2$, and the loop is of length $k \geq 4$. These parameters have been carefully chosen to closely resemble the real-world scenarios encountered in biochemical processes, enhancing the practical relevance of our study.

1. Introduction

The rapid advancement of information technology and the widespread use of social networking have led to an exponential surge in data generation [1]. In response to this challenge, the field of Deoxyribonucleic Acid (DNA) storage has emerged, leveraging the progress in DNA synthesis and sequencing technologies to serve as a promising and ideal medium for storing vast amounts of digital information [2]. However, it is essential to acknowledge that the methods for synthesizing and sequencing DNA sequences are far from perfect. If the codewords are not chosen appropriately, unintended (non-selective) hybridization or errors may occur during the process. To address these potential issues and ensure the reliability of DNA storage, numerous exceptional error correction methods [3–5] and DNA clustering methods [6,7] have been proposed, which play a crucial role in enhancing the accuracy and efficiency of DNA storage systems, mitigating errors, and ensuring the integrity of the stored digital information.

In conventional DNA storage approaches, the utilization of short single-stranded DNA sequences, also referred to as oligonucleotide sequences, is common. Each of these sequences represents an oriented word, comprising four distinct nucleotide bases: Adenine (A), Thymine (T), Cytosine (C), and Guanine (G). The Watson-Crick complementarity relationships are denoted as follows: $\overline{T} = A$, $\overline{A} = T$, $\overline{C} = G$, and $\overline{G} = C$. These complementary base pairings play a fundamental role in DNA replication, transcription, and other essential biological processes.

A secondary structure in the context of DNA storage refers to the existence of two non-overlapping subsequences that are reverse complements of each other. This structural arrangement causes the sequence to fold back onto itself through a complementary base pair, resulting in the formation of a stem-loop structure as shown in Fig. 1. Such secondary structures have been identified as a significant source of potential irreparable read-out failures within DNA storage systems [8]. For example, a large number of read-out failures in the DNA storage system described in [9] was attributed to the formation of hairpins, a special secondary structure formed by oligonucleotide sequences. It is necessary to stress the fact that DNA code design must take secondary structure considerations into account [10]. In the field of DNA sequence analysis, some researchers have presented rigorous solutions to avoid secondary structures in DNA codes, both for significantly large [11] and small [12] stem lengths. However, as of yet, no efficient method has been devised to address DNA sequences with realistic and appropriately sized stems. This represents a significant gap in the current state of research, leaving a critical aspect of DNA analysis unexplored.

In this study, we focus on the construction of DNA codes with a specified length n, aiming to avoid the formation of secondary structures with stem lengths equal to or greater than a given integer $m \geq 2$.

* Corresponding author.
E-mail addresses: nessajzhang@mail.nankai.edu.cn (R. Zhang), whming@tju.edu.cn (H. Wu).

https://doi.org/10.1016/j.csbj.2023.11.035
Received 23 September 2023; Received in revised form 16 November 2023; Accepted 17 November 2023
Available online 29 November 2023

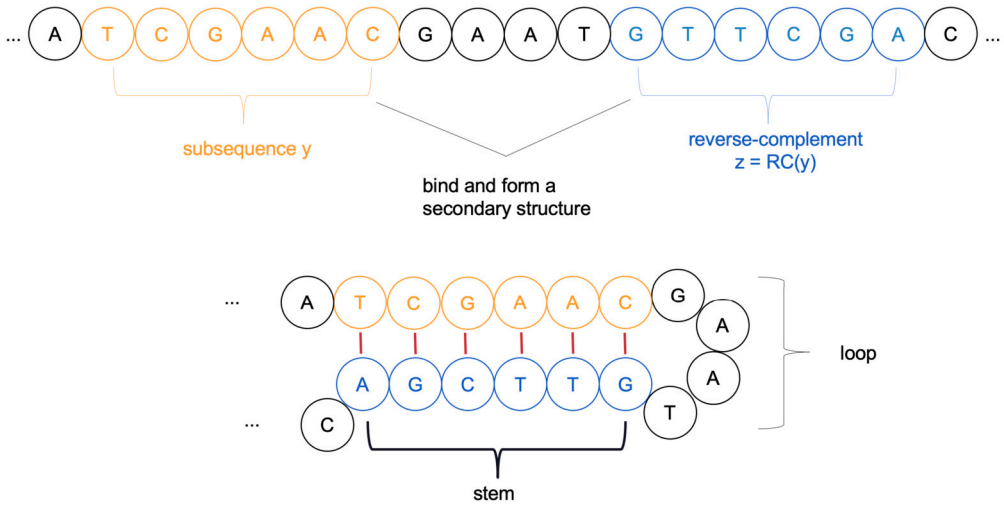

Fig. 1. DNA secondary structure model.

It is crucial to note that we refrain from categorizing m as either excessively large or small, or limiting the consideration to odd stems. In real-world applications of DNA storage, biochemists typically address the secondary structure problem with a stem length of $m \geq 6$ and a loop length of at least 4 [12]. As depicted in Fig. 1, the subsequences "TCGAAC" and "AGCTTG" form a secondary structure. One of these sequences, either "TCGAAC" or "AGCTTG", is referred to as a stem, and its length is indicated as "m", while the sequence between them is termed a loop, with its length denoted as "k". If the stem length m is chosen to be too large, such as $m \geq 3\log_2 n + 4$ as addressed in [11], several shorter secondary structures may still form, leading to the emergence of uncorrectable errors in the DNA sequence. On the other hand, selecting m to be too small, for instance, 2 or 3 as demonstrated in [12], may cause relatively short palindromic sequences that do not form a secondary structure, thereby requiring no further processing.

The main contributions of this paper are three-fold:

- We propose a novel secondary structure-avoidance code tailored for stems of length $m \geq 2\log_2 n + 2$ and loops of length $k \geq 4$ within a specific biological context. Notably, under certain conditions, this code can be extended to accommodate stems of length $m \geq \log_2 n + 2$.
- We put forth the definition of (m, k)-SSA sequence, along with an associated theorem that incorporates the length of the loop denoted by "k".
- Numerical results indicate the relationship between the length and quantity of the loops.

2. Preliminary

In this study, we represent the DNA alphabet set as $\mathcal{D} = \{A, T, C, G\}$. Given any two sequences $\mathbf{x}$ and $\mathbf{y}$, we denote $\mathbf{xy}$ as the concatenation of the two sequences. For a given sequence $\mathbf{x}$ of length n, we define a consecutive subsequence $\mathbf{y}$ of length l as $\mathbf{y} = x_i x_{i+1} \dots x_{i+l-1}$, where $\mathbf{y}$ consists of consecutive elements from $\mathbf{x}$. Furthermore, if we have two subsequences $\mathbf{y} = x_i x_{i+1} \dots x_{i+l-1}$ and $\mathbf{z} = x_j x_{j+1} \dots x_{j+s-1}$ from sequence $\mathbf{x}$, we categorize $\mathbf{y}$ and $\mathbf{z}$ as non-overlapping if either of the following conditions is satisfied: (i) $i + l - 1 < j$, or (ii) $i > j + s - 1$. In other words, for subsequences to be non-overlapping, there must be a clear separation between their respective starting and ending positions in the original sequence $\mathbf{x}$.

We investigate the construction of DNA codes with a codeword length denoted by n, along with two specified integers, namely $m \geq 2\log_2 n + 2$ and $k \geq 4$. The primary objective is to design DNA codes that effectively avoid the formation of secondary structures possessing stem lengths equal to m, as well as loops of a size greater than or equal to k. In the context of DNA oligonucleotides, it is common for these molecules to exhibit limited flexibility and are unlikely to form sharp turns or bend over short regions. Consequently, we introduce a slight refinement to the condition for folding by stipulating that the reverse-complement subsequences must maintain a sufficiently large separation between them.

Definition 1. For a DNA sequence $\mathbf{x} \in \mathcal{D}^n$, let $\mathbf{y} = x_i x_{i+1} \dots x_{i+l-1}$ and $\mathbf{z} = x_j x_{j+1} \dots x_{j+s-1}$ represent two non-overlapping subsequences of $\mathbf{x}$. We define $\mathbf{y}$ and $\mathbf{z}$ as reverse-complement, denoted as $\mathbf{y} = RC(\mathbf{z})$, when $x_{i+t} = \overline{x}_{j+s-1-t}$ for every $t \in [0, s-1]$.

Definition 2. Given integers $2 < m \leq n$, a DNA sequence $\mathbf{x} \in \mathcal{D}^n$ is said to be an m-secondary structure avoidance sequence (abbreviated as an m-SSA sequence), if it does not contain any two non-overlapping reverse-complement subsequences of length m. A code $C \subseteq \mathcal{D}^n$ is called an m-SSA code if every codeword in C is an m-SSA sequence.

Unfortunately, the aforementioned studies [13,11,12,14] have not comprehensively addressed the influence of the loop size k on the formation of secondary structures. This limitation arises from the recognition that biochemical self-hybridization differs from the simplified sequence encoding used in these studies. Notably, DNA oligonucleotides typically exhibit a lack of sharp turns or bending over small regions, making the inclusion of the parameter k particularly pertinent and meaningful in this context.

3. (m, k)-SSA sequence

In our work, we extend the concept of m-SSA to the more natural concept of (m, k)-SSA, we can use data simulation to analyze how many reverse-complements that are too close which have a very small probability of forming a secondary structure and do not need to be avoided.

Definition 3. For a DNA sequence $\mathbf{x} \in \mathcal{D}^n$ with $\mathbf{y} = x_i x_{i+1} \dots x_{i+l-1}$ and $\mathbf{z} = x_j x_{j+1} \dots x_{j+s-1}$, where $j > i + l - 1$, are two non-overlapping subsequences of $\mathbf{x}$. We denote $d(\mathbf{y}, \mathbf{z}) \triangleq j - i - l + 1$ as the "distance" between $\mathbf{y}$ and $\mathbf{z}$. Moreover, if $\mathbf{y} = RC(\mathbf{z})$, $k = d(\mathbf{y}, \mathbf{z})$ is the length of the loop of the secondary structure of this reverse-complement.

Definition 4. A DNA sequence $\mathbf{x} \in \mathcal{D}^n$ is said to be an (m, k)-SSA sequence if, for every pair of subsequences y and z within x, where $y = RC(z)$, the following condition holds: either the length of y is less

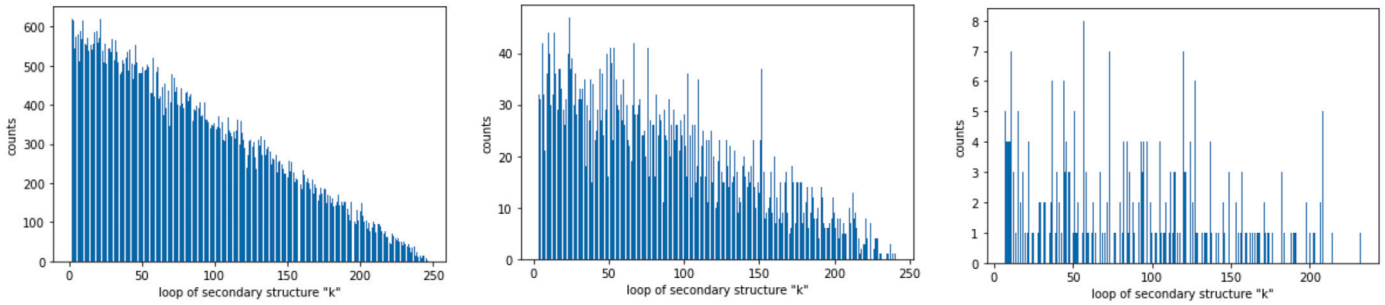

Fig. 2. The relationship between the length and quantity of the loops of reverse-complements with different m in 10,000 sequences.

than m (i.e., $length(y) < m$), or the distance between them is less than k (i.e., $d(y,z) < k$). A code $\mathcal{C} \subseteq \mathcal{D}^n$ is referred to as an (m,k)-SSA code when each codeword within $\mathcal{C}$ satisfies the criteria of being an (m,k)-SSA sequence.

Remark 1. Our definition (m,k)-SSA is much "weaker" than m-SSA in [11] and [12], that is to say, not all (m,k)-SSA sequences are m-SSA, while all m-SSA sequences have the property of (m,k)-SSA. Since we notice that not all reverse-complements form a secondary structure, A DNA oligonucleotide usually does not bend over small regions [15], it would be more natural to have a coding theoretic solution considering the length of loop "k".

As shown in Fig. 2, our data, which comprises DNA sequences randomly generated without consideration of GC content and homopolymer avoidance, simulates the relationship between the length of the sequence n and the number of loops when m is set to 6, 8 and 10, respectively. As m increases, the probability of finding a corresponding reverse complement decreases significantly, and the number of k is significantly reduced. It's worth noting that a substantial proportion of the DNA sequences we analyzed consist of short loops in their secondary structure, with approximately twenty-five percent of all sequences having loops shorter than 20. The presence of these short loop sequences suggests that they are not easily forming secondary structures, implying that it may be appropriate to retain these reverse-complements in the sequences.

Definition 5. For a positive integer N, the DNA-representation of N is the replacement of symbols in the quaternary representation of N over $D = \{0,1,2,3\}$ by following rule: $0 \leftrightarrow A$, $1 \leftrightarrow T$, $2 \leftrightarrow C$, and $3 \leftrightarrow G$.

Theorem 1. *Given m, n and $k > 0$, if a sequence $\mathbf{x} \in \mathcal{D}^n$ is (m,k)-SSA, then $\mathbf{x}$ is (m',k')-SSA, for all $m' \geq m, k' \geq k$.*

Proof. We establish this theorem through a proof by contradiction. Suppose there exists values $m' \geq m$ and $k' \geq k$ for which $\mathbf{x}$ is not (m',k')-SSA. In such a case, there must be a pair of subsequences $\mathbf{y} = RC(\mathbf{z})$ with a length of m' and $d(\mathbf{y},\mathbf{z}) \geq k' \geq k$. Now, since the lengths of both $\mathbf{y}$ and $\mathbf{z}$ are $m' > m$, it follows that $\mathbf{y}$ must contain a subsequence of length m denoted as $\mathbf{y}'$, and $\mathbf{z}$ must have a subsequence of length m as well, denoted as $\mathbf{z}'$. Importantly, $d(\mathbf{y}',\mathbf{z}') \geq d(\mathbf{y},\mathbf{z}) \geq k$, signifying that $\mathbf{y}'$ and $\mathbf{z}'$ are reverse-complements with a length of m and a loop of length greater than or equal to k. Consequently, it implies that $\mathbf{x}$ is not (m,k)-SSA, leading to a contradiction. □

4. Constructions of (m,k)-SSA codes for $m \geq 2\log_2 n + 2$

Given the conditions: $n > m > 2$, $n > 2^6$, $m \geq 2\log_2 n + 2$, and $k \geq 4$, we establish a parameter $t = 0.5m = \log_2 n + 1$ for later reference. Additionally, let $u \in \mathcal{D}^{n-1}$ represent the message DNA sequence, which is the original sequence before encoding.

4.1. Encoder

Our encoding algorithm can be simply divided into three steps to execute sequentially:

- **Step 1** (prefix): In the encoding process, we initiate by adding the nucleotide T as a prefix to the message DNA sequence $\mathbf{u} \in \mathcal{D}^{n-1}$, resulting in a sequence $\mathbf{x} \in \mathcal{D}^n = Tu$. Subsequently, we assess whether the sequence $\mathbf{x}$ is already an (m,k)-SSA DNA sequence. If it satisfies the (m,k)-SSA criteria, the encoder directly outputs the sequence $\mathbf{x}$, otherwise, it proceeds to the next step in the encoding process. For more specific information, please refer to Algorithm 1.
- **Step 2** (replacing): Next, the encoder conducts a left-to-right scan of the entire sequence, aiming to identify the earliest occurrence of a pair of non-overlapping subsequences, denoted as $\mathbf{y}$ and $\mathbf{z}$, each with a length of t, that meet the condition $\mathbf{z} = RC(\mathbf{y})$. Here, $\mathbf{y}$ initiates at position i, $\mathbf{z}$ begins at position j, and the separation between these starting positions, denoted as $j - i$, must be greater than or equal to $t + k$. Alternatively, the encoder also looks for the initial occurrence of a subsequence $\mathbf{s}$ within the input $\mathbf{x}$ that conforms to the pattern $\mathbf{s} = (x_1x_2)^{t/2}$, where $x_1, x_2 \in D$.
  - **Case 1**: If it finds a pair of reverse-complement subsequences $\mathbf{y}$ and $\mathbf{z}$ of $\mathbf{x}$, we can write $\mathbf{x} = X_1\mathbf{y}X_2\mathbf{z}X_3$, where X_1, X_2 and X_3 are all subsequences of $\mathbf{x}$. We solve this problem with so-called "C"-replacement as follows:

$$X_1yX_2zX_3 \xrightarrow{j-i-t\geq k} X_1yX_2X_3 \rightarrow CK\alpha X_1yX_2X_3 \tag{1}$$

When multiple secondary structures exist, repeating this process is straightforward, as outlined in Eq. (2). In this case, we have two pairs of reverse-complements: y_1 and z_1, as well as y_2 and z_2. These pairs are defined with y_1 and z_1 starting at positions i_1 and j_1, and y_2 and z_2 starting at positions i_2 and j_2.

$$X_1y_1X_3y_2X_3z_1X_4z_2X_5 \xrightarrow[j_1-i_1-t\geq k]{j_2-i_2-t\geq k} X_1y_1X_2y_2X_3X_4X_5 \rightarrow$$
$$CK_2\alpha_2CK_1\alpha_1X_1y_1X_2y_2X_3X_4X_5 \tag{2}$$

The encoder establishes a pointer denoted as $P = CK\alpha$. This pointer functions as a prefix that encodes information about the loop size and the reverse-complement position. Specifically, C represents a single nucleotide that denotes the reverse complement, K stands for the DNA-encoded representation of the length of loop k, and α represents the DNA-encoded representation of the position i at which the sequence $\mathbf{y}$ begins. It is worth noting that we define the starting index of $\mathbf{z}$ as j. Both K and i have a length of $\log_4 n$. Consequently, the pointer P is of a length of $1 + 2\log_4 n = 1 + \log_2 n$. Following this, the encoder proceeds to

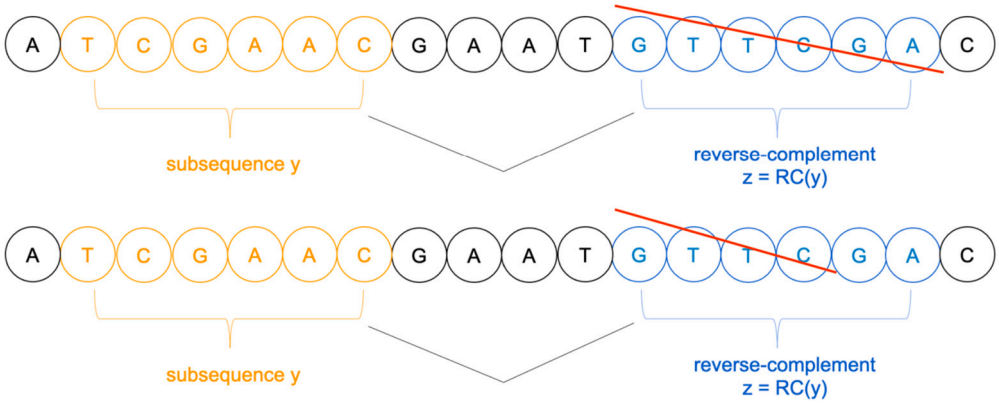

Fig. 3. While previous work removes the entire **z** of length 6 when the encoder finds the reverse-complement of length $l > 6 > m = 4$, our work removes exactly the $m = 4$ sign and no secondary structure is formed.

remove the sequence **z** from **x** and adds the prefix P to the beginning of **x**.

Remark 2. The subsequence **z** that has been removed has a length of $t = 0.5m = 1 + \log_2 n$. The pointer P that has been added at the beginning also has a length of $1 + \log_2 n$. Notably, if $m \geq 2 + 2\log_2 n$, the length of the insertion pointer P is either less than or equal to the length of the subsequence **z** that was removed. Thus, this specific method of sequence replacement, which we refer to as "C"-replacement, does not result in an increase in the overall length of the sequence **x**.

– **Case 2**: If the encoder finds the subsequence **s** of **x** of the form $\mathbf{s} = (x_1x_2)^{t/2}$, where $x_1, x_2 \in D$. We can write $\mathbf{x} = X_1\mathbf{s}X_2 = X_1(x_1x_2)^{t/2}X_2$. In particular, we solve this problem with a so-called "A"-replacement as follows:

$$X_1(x_1x_2)^{t/2}X_2 \to X_1X_2 \to Ax_1x_2\beta X_1X_2. \tag{3}$$

The encoder sets a pointer $Q = Ax_1x_2\beta$, where A as a single nucleotide stands for the repeated patterns of size 2, x_1x_2 stands for the repeated words x_1 and x_2, and β stands for the DNA-representation of i which is the position where **s** starts.

Remark 3. Given that β has a length of $\log_4 n$, the pointer Q is $1 + 2 + \log_4 n = 3 + \frac{1}{2}\log_2 n$ in length. Consequently, when we calculate $1 + \log_2 n - (3 + 0.5\log_2 n)$, it results in $\frac{1}{2}\log_2 n - 2$. Importantly, this quantity is greater than 1 when $n > 2^6$. Therefore, the replacement reduces the length of the sequence **x** by more than one symbol, specifically $\frac{1}{2}\log_2 n - 2$, when n exceeds 64 (i.e., $n > 2^6$).

- **Step 3** (adding suffix): The encoder repeats these scanning and sequence replacing steps until the sequence **x** no longer contains any reverse-complement pairs, and $\mathbf{s} = (x_1x_2)^{t/2}$ has a length greater than or equal to $t = 0.5m$. Importantly, as our sequence replacement method consistently reduces the length of the sequence, this process is guaranteed to eventually terminate. In our sequence replacement strategy, the encoder aims to reduce the length of the current sequence **x** while preserving its (m, k)-SSA properties. When the length of the current sequence is denoted as n', where $n' < n$, the encoder appends a suffix of length $n_s = n - n'$ to extend the sequence to the desired length n.

In selecting unpaired nucleotides as a suffix, we consider the potential benefits of maintaining a GC content closer to 50%. To ensure that the generated suffix complies with the (m, k)-SSA condition, preserving the integrity and effectiveness of our output codeword, we employ a suitable and efficient method as follows:

- If the length of the suffix n_s is even, we append $\mathbf{r} = (GT)^{N_1/2}$ to the end of the current sequence **x**.
- If the length of the suffix n_s is odd, we append $\mathbf{r} = (GT)^{N_1-1/2}G$ to the end of the current sequence **x**.

Remark 4. In our work, we only delete part of the whole subsequence that is exactly m in length, and keep the rest (see Fig. 3), because they are not long enough to form a secondary structure (see Fig. 4). The reduced length of the pruned subsequence helps to reduce the number of operations and improve the success rate of decoding, and makes it possible to handle sizes of m that are closer to biochemical needs $(2\log_2 n + 2)$. Even if our operations form new secondary structures or repeated patterns, since our algorithm keeps repeating, eventually all (old, new) secondary structures and repeated patterns of size 2 will be processed.

We denote the encoder as Enc that $\mathbf{x} = \mathbf{Enc}(\mathbf{u})$, when **x** is the sequence after **u** is encoded.

Theorem 2. $\mathbf{x} = \mathbf{Enc}(\mathbf{u})$ *is* (m, k)*-SSA for all* $\mathbf{u} \in \mathcal{D}^{n-1}$.

Proof. Suppose $\mathbf{x} = \mathbf{x_1x_2} = \mathbf{Enc}(\mathbf{u}) \in \mathcal{D}^n$ is the output of our encoder, where $\mathbf{x_1}$ and $\mathbf{x_2}$ are subsequences of **x**. $\mathbf{x_1}$ is (t, k)-SSA and the length of the repeated patterns (mentioned in the step 3 in Encoder algorithm) of size 2 in $\mathbf{x_1}$ is of length at most $t = \log_2 n + 1$, and $\mathbf{x_2}$ is the generated suffix of $\mathbf{x_1}$ at the suffix adding phase. Consider an arbitrary subsequence of length m, $\mathbf{y} = \mathbf{y_1y_2}$, where $\mathbf{y_1}$ is the subsequence of $\mathbf{x_1}$ and $\mathbf{y_2}$ is the subsequence of $\mathbf{x_2}$.

As depicted in Fig. 5, we consider the following cases due to possible different positions of y in x as follows:

- **Case 1**: If $\mathbf{y_1}$ is of length more than or equal to $t = 0.5m$ (particularly including the case that $\mathbf{y_1}$ is of length m which means **y** is totally a subsequence of $\mathbf{x_1}$), since $\mathbf{x_1}$ is (t, k)-SSA, $\mathbf{y_1}$ cannot find its reverse-complement whose loop is of length more than or equal to k and stem of length more than or equal to t in $\mathbf{x_1}$, and since $\mathbf{x_2}$ is of the form "$GTGT\ldots$", there are no such repeated patterns of size two of length more than or equal to t in $\mathbf{X_1}$ due to the construction of our encoder, the same for $\mathbf{y_1}$ because $\mathbf{y_1}$ is its subsequence. Thus, $\mathbf{y_1}$ cannot find its reverse-complement in both $\mathbf{x_1}$ and $\mathbf{x_2}$.
- **Case 2**: If the length of $\mathbf{y_1}$ is less than $t = 0.5m$ (partically including the case where $\mathbf{y_1}$ has length 0, indicating that **y** completely constitutes a subsequence of $\mathbf{x_2}$). Simultaneously, the length of $\mathbf{y_2}$,

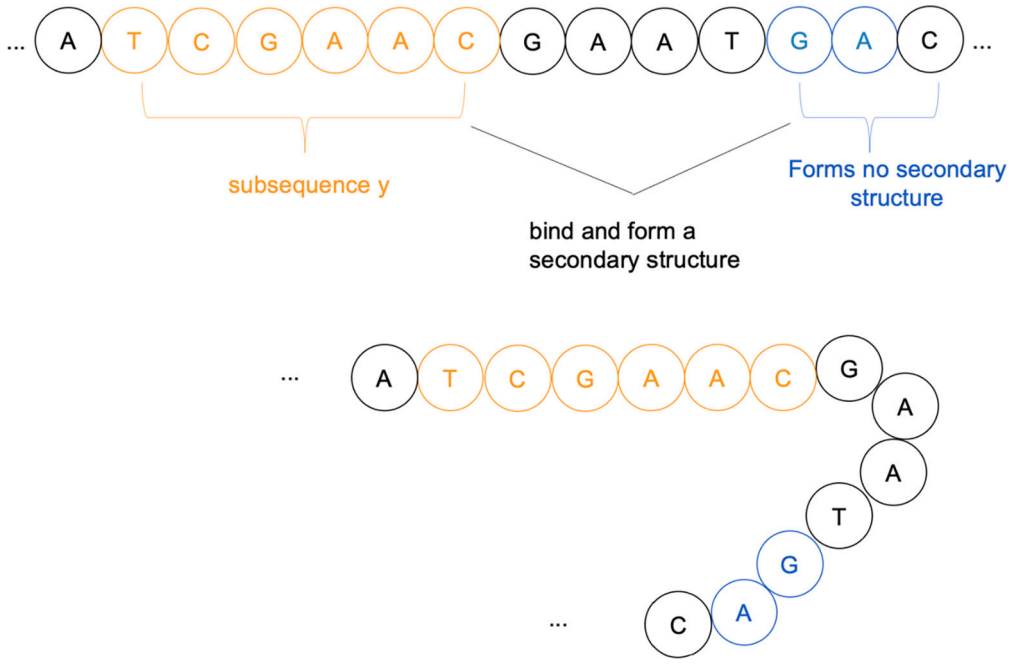

Fig. 4. The removal of shorter sequences that do not form secondary structures.

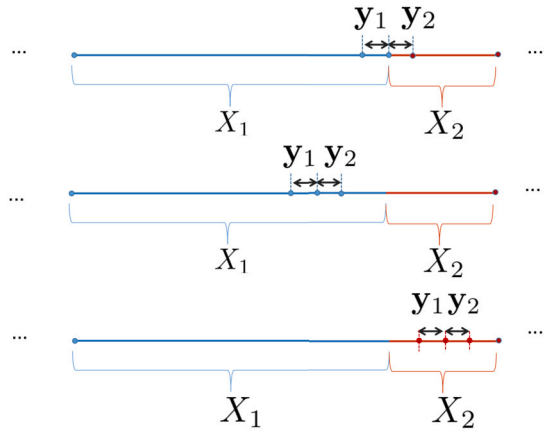

Fig. 5. Positions of $\mathbf{y}$ in $\mathbf{x}$.

which consists of repetitive patterns of size 2, exceeds t, leading to the inability to identify its reverse-complement within $\mathbf{x_1}$ due to the constraints of the encoder's design. Furthermore, the reverse-complement of $\mathbf{y_2}$ cannot be located within $\mathbf{x_2}$ either, as instances such as "$GTGT\ldots$" do not possess a viable reverse-complement, where T and G do not conform to the Watson-Crick complementarities. □

Algorithm 1: Framework of m-SSA Encoding.

Input: Message DNA sequence: $\mathbf{u}$; Stem length: m; Loop length: k.
Output: (m,k)-SSA codeword: $\mathbf{x}$.

$\mathbf{x} = T\mathbf{u}$
while $\mathbf{x}$ *is not* (m,k)*-SSA* **do**
  while $\mathbf{x}$ *has a reverse-complement pair of subsequence* **do**
    "C"-replacement
  while $\mathbf{x}$ *has subsequence* $\mathbf{s} = (x_1x_2)^{(t/2)}$ **do**
    "A"-replacement
  $\mathbf{x}$ adds the suffix $GT\ldots$
return $\mathbf{x}$

4.2. Decoder

Given a received DNA sequence $\mathbf{x}$ with a length of n, our decoding procedure involves a sequential scan of the sequence from left to right. During this scanning process, the decoder examines the first symbol in the sequence.

- **Case 1**: If the first symbol corresponds to "T", it serves as a signal that the subsequent $n-1$ symbols collectively form a (m,k)-SSA sequence. Under this circumstance, the decoder promptly removes the terminal symbol T, and proceeds to identify the ensuing $n-1$ symbols as the message DNA sequence. This identified message DNA sequence is then extracted and outputted by the decoder.
- **Case 2**: If the first symbol is "C", the decoder scans from left to right and takes the prefix of length $1+\log_2 n$ as the pointer P, which is prepended to the sequence after the "C"-replacement. We consider the pointer P to be of the form $CK\alpha$. The decoder calculates the positive integers whose DNA representation is K, and α, and sets k and i to be these two integers, they are the length of the loop of secondary structure and the start of $\mathbf{y}$ which is $RC(\mathbf{z})$. Thus, the decoder removes the pointer $P = CK\alpha$ and inserts $\mathbf{z} = RC(\mathbf{y})$ to sequence $\mathbf{x}$ at index $i+k+m'$.

$$CK\alpha X_1 yX_2X_3 \to X_1 yX_2X_3 \to X_1 yX_2zX_3. \tag{4}$$

- **Case 3**: If the received sequence starts with A, the decoder scans from left to right and takes the prefix of length $3+0.5\log_2 n$ as the pointer Q, which is prepended to the sequence after the "C"-replacement. We consider the pointer Q to be of the form $Ax_1x_2\beta$, where x_1 and x_2 are the two symbols in the repeated patterns of size 2 and β is the DNA representation of where the repeated patterns "$(x_1x_2)^{t/2}$" was. The decoder calculates the positive integer "j" whose DNA-representation is β, and then removes the pointer $Q = Ax_1x_2\beta$ and inserts $(x_1x_2)^{t/2}$ to $\mathbf{x}$ at index j.

$$Ax_1x_2\beta X_1X_2 \to X_1X_2 \to X_1(x_1x_2)^{t/2}X_2. \tag{5}$$

- **Case 4**: In the event that the received sequence initiates with the nucleotide G, which deviates from our intended design expectations, the decoder promptly identifies and reports errors. Subsequently, the decoder leverages error-correcting codes to rectify these errors.

The decoding process concludes when the initial symbol in the sequence corresponds to the symbol "T", which serves as a terminal marker. At this point, the decoder considers the subsequent $(n-1)$ symbols in the sequence, disregarding any additional symbols and suffixes that might have been introduced during the encoding stage. These $(n-1)$ symbols are then recognized and treated as the message data, containing the essential information intended for extraction and interpretation from the decoded sequence.

Remark 5. If the length of the cycle k is given or has already been traversed during the search for a palindrome sequence, then it eliminates the necessity of storing this value in the pointer. Consequently, the pointer representation, initially expressed as $P = CK\alpha$, can be simplified to $P = C\alpha$. This simplification does not compromise the generality of our conclusion. Therefore, our findings can be extended to situations where m satisfies $m \geq \log_2 n + 2$. This condition aligns more closely with the practical requirements of biochemical experiments, providing greater relevance and applicability to real-world scenarios.

4.3. Numerical results

Algorithm 2: Calculation of RC pairs reduced rate.

Input: Length of message DNA sequence: 2^n; Stem length: $m \geq 2\log_2 n + 2$; Loop length: k.
Output: RC pairs reduced rate: $(R - R')/R$.

$i \leftarrow 0$
while $i \leq 10,000$ **do**
  Randomly generate a 2^n long DNA sequence.
  Scan and count the number R of RC pairs.
  Encode the DNA sequence.
  Scan and count the number R' of RC pairs again.
  $i = i + 1$
return $(R - R')/R$

We devised an experiment to determine the count of reverse-complementary sequence pairs with a length of m within randomly generated DNA sequences, while varying the values of n and adjusting the values of m accordingly.

We randomly generated a total of 60,000 DNA sequences of varying lengths (128, 256, 512, 1024, 2048, and 4096) with 10,000 sequences for each length category. We then quantified the count of reverse-complementary sequences under different values of m. In addition, for the scenario where $m = 2\log_2 n + 2$, as explored in our study, we applied our encoding scheme to represent the original random DNA sequences. The encoded sequences obtained through this method exhibit the property of being (m, k)-SSA. Our findings reveal that when m is significantly large (such as $3\log_2 n + 4$ in [11]), numerical experiments suggest that it becomes exceedingly improbable to identify such lengthy pairs of reverse-complementary subsequences within a DNA sequence. However, when m is appropriately adjusted to a smaller but still substantial value, our encoding scheme proves to be effective, as illustrated in Fig. 8. Compared with [11], our work can deal with the secondary structure caused by the reverse-complementary sequences with smaller values of m, as shown in Fig. 6.

The relationship between n, m, and k is illustrated in Fig. 7, where we select the top 25 values and the top 20% after sorting k in ascending order. When the stem length $m \geq 2\log_2 n + 2$, it becomes highly probable that reverse-complementary sequences will emerge. The length of these reverse-complementary sequences significantly influences the stability of DNA sequences.

In Fig. 8, our findings demonstrate that when $m = 3\log_2 n + 4$, as described in [11], DNA sequences exhibit minimal formation of extended reverse complementary pairs (as indicated by the blue line, which closely aligns with the x-axis). In such cases, there is no pressing need for specialized encoding and decoding methods. Conversely, when

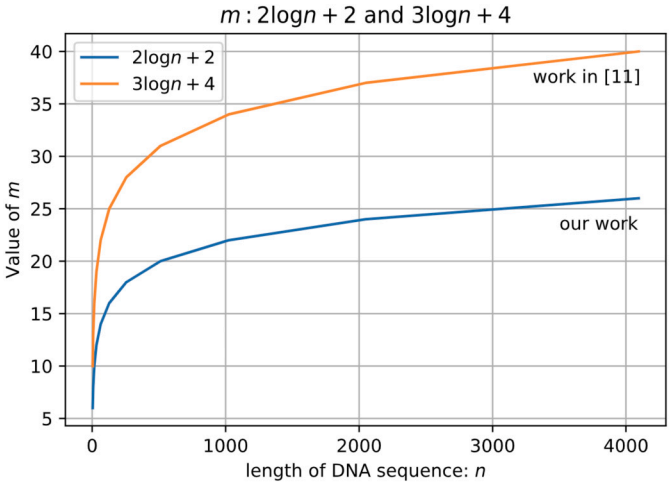

Fig. 6. The stem length m that both algorithms can handle.

$m = 2\log_2 n + 2$, the count of potential reverse complementary pairs significantly increases (as seen by the orange line). However, our encoding approach effectively circumvents these reverse complementary pairings (as represented by the dashed red line).

Next, we conducted numerical experiments using Algorithm 2 to quantify the reduction of reverse-complement pairs in the original DNA sequence achieved through our encoding process. We compared the count of these pairs before and after encoding. The experimental results, as visualized in Fig. 8, clearly demonstrate the effectiveness of our encoding method in eliminating all reverse-complement pairs with a "distance" greater than k. This is because the count R' of reverse-complement pairs is reduced to zero after encoding. Essentially, the sequences after encoding are all (m, k)-SSA, and the reduction rate of reverse-complement pairs, quantified as $(R - R')/R$, is equal to 1.

4.4. Homopolymer avoiding analysis

Reading and writing of a nucleotide at consecutive positions are one of the significant causes of insertion and deletion errors [16,17]. A homopolymer of run-length r is a DNA string with a sub-string of length r, where all nucleotides of the sub-string are the same. For example, the DNA string CGGGGGATTC has a sub-string GGGGG, and therefore, has a homopolymer of run-length five.

Unfortunately, in our algorithm, adding prefixes and deleting subsequences may both generate homopolymers. How to reduce or even avoid the influence of homopolymers is an issue that must be considered, and we take the homopolymer of run-length five into account.

As said above, adding a prefix $CK\alpha$ or deleting subsequence z may generate homopolymers, we take $n = 1024$ for example:

- Case 1: Adding prefix $CK\alpha$. As mentioned above, C is an individual base, K and α are both of length $\log_4 n$, that is, of length 5. We can calculate the probability of generating new homopolymers on a case-by-case basis, and the result is: $\frac{4+3+4+3*4*\frac{1}{4}*2+4*3*4*\frac{4}{4*4}*2}{4^5} = \frac{41}{1024}$.
- Case 2: Deleting z. We can also calculate the probability when X_2 and X_3 create new homopolymer: $\frac{3*4*\frac{1}{4}*2+4*3*4*\frac{4}{4*4}*2}{4^5} = \frac{30}{1024}$.

As a result, the probability of generating new homopolymers is $\frac{71}{1024}$ when there is a secondary structure, which is rather small, which means when the encoding finds an encoded sequence with homopolymers, we can recode until no homopolymers appear.

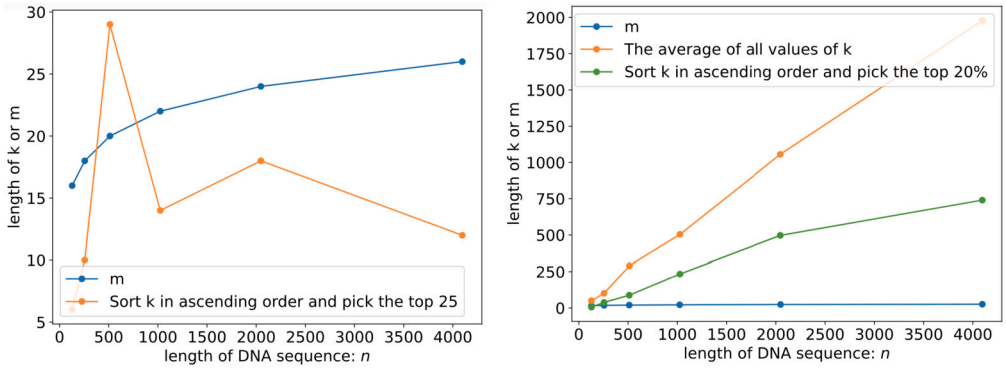

Fig. 7. The relationship between n, m and k.

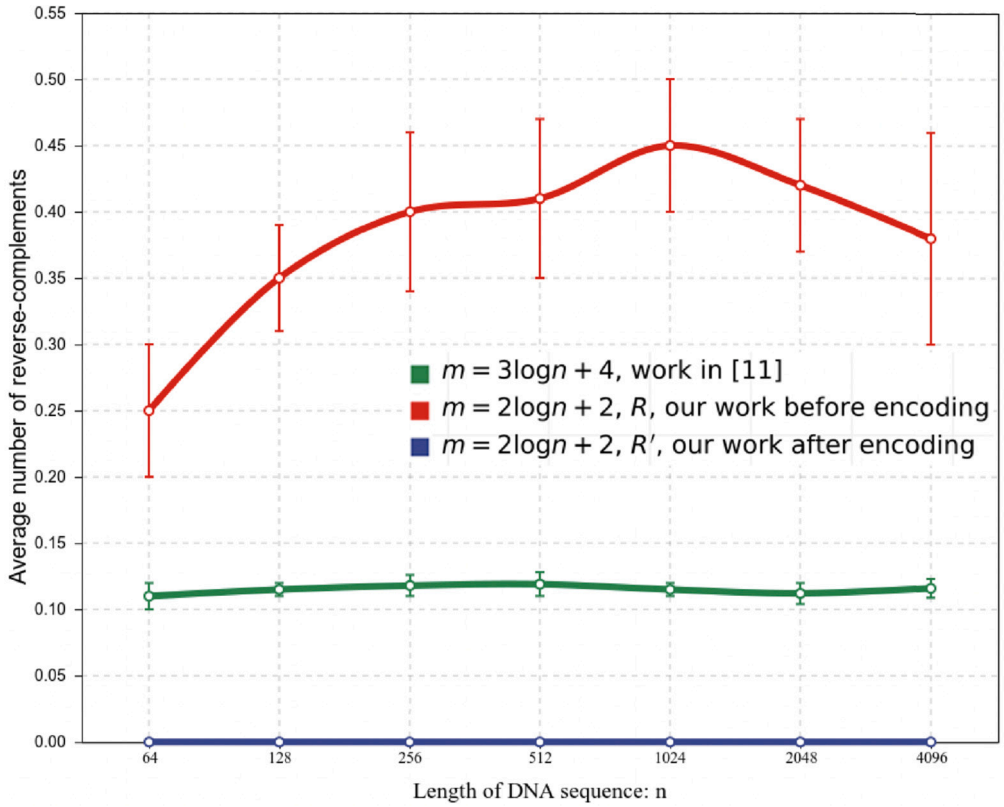

Fig. 8. Average number of reverse-complements under different values of both n and m.

4.5. Application to short DNA strings

Presently, synthesis technology falls short in generating lengthy DNA strands, with current technologies limited to producing oligonucleotides of a maximum length of 300. It is pragmatic to conduct research within the confines of existing science and technology. The proof of Theorem 2, validating the correctness of our algorithm, is independent of the selection of n. Whether n is small (e.g., around 100-200 base pairs) or large (up to 4,000 base pairs), the same algorithm is applicable. It's worth highlighting that our method also outperforms existing approaches in the context of contemporary DNA synthesis technologies, which often involve shorter sequences (e.g., around 100-200 base pairs), as depicted in Fig. 8. Importantly, our work demonstrates versatility by effectively accommodating short DNA strings as well. This adaptability underscores the extensive range of applications for our algorithm.

4.6. Complexity analysis

The time complexity of both the encoder and the corresponding decoder for a codeword of length n is linear in n. This is due to the fact that the number of sequences replacing operations during the encoding process is at most $n-m$, which is represented as $\mathcal{O}(n)$. Additionally, each replacing operation, which includes steps like prepending a prefix or converting the quaternary representation to the DNA representation of an integer, has a constant time complexity of $\mathcal{O}(1)$.

As a result, the overall time complexity of the encoder and decoder operations remains linear with respect to the length of the codeword n. This linear complexity is advantageous as it ensures efficient and manageable processing times for DNA code construction and decoding, making it suitable for handling large-scale data storage and retrieval tasks.

5. Conclusion

In this study, we have proposed a novel approach to tackle the challenge of avoiding secondary structure formation during DNA storage. Additionally, we have put forth a new definition of (m, k)-SSA to describe the secondary structure. Our method has demonstrated its effectiveness, particularly for moderately large values of the parameter m. Meanwhile, for stem lengths $m \geq 2\log_2 n + 2$, we have introduced an algorithm that incorporates a single redundant codeword, thereby achieving linear complexity in the transformation of regular codewords into secondary structure-avoiding codes. Compared to previous algorithms, our proposed method exhibits broader applicability and lower computational complexity. In future work, we aim to reduce the large value of m and incorporate error correction codewords into the current algorithm.

CRediT authorship contribution statement

Rui Zhang: Conceptualization, Methodology, Software, Writing – original draft. **Huaming Wu:** Investigation, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This work is supported by the National Key R&D Program of China under Grant Number 2020YFA0712100 and the National Natural Science Foundation of China under Grant Number 62071327.

References

[1] Dong Yiming, Sun Fajia, Ping Zhi, Ouyang Qi, Qian Long. Dna storage: research landscape and future prospects. Nat Sci Rev 2020;7(6):1092–107.
[2] Hao Yaya, Li Qian, Fan Chunhai, Wang Fei. Data storage based on dna. Small Struct 2021;2(2):2000046.
[3] Yan Zihui, Liang Cong, Wu Huaming. A segmented-edit error-correcting code with re-synchronization function for dna-based storage systems. IEEE Trans Emerg Topics Comput 2022:1–13.
[4] Organick L, Ang SD, Chen YJ, Lopez R, Yekhanin S, Makarychev K, et al. Random access in large-scale DNA data storage. Nat Biotechnol 2018;36(3).
[5] Limbachiya Dixita, Benerjee Krishna Gopal, Rao Bansari, Gupta Manish K. On dna codes using the ring z4 + wz4. In: Proc. IEEE int. symp. inf. theory; 2018. p. 2401–5.
[6] Wang Penghao, Cao Ben, Ma Tao, Wang Bin, Zhang Qiang, Zheng Pan. DUHI: dynamically updated hash index clustering method for DNA storage. Comput Biol Med Sep. 2023;164:107244.
[7] Qu Guanjin, Yan Zihui, Wu Huaming. Clover: tree structure-based efficient DNA clustering for DNA-based data storage. Brief Bioinform Aug. 2022;23(5).
[8] Benerjee Krishna Gopal, Banerjee Adrish. On homopolymers and secondary structures avoiding, reversible, reversible-complement and gc-balanced dna codes. In: Proc. IEEE int. symp. inf. theory. Espoo, Finland: IEEE; 2022. p. 204–9.
[9] Mansuripur Masud, Khulbe Pramod K, Kuebler Stephen M, Perry Joseph W, Giridhar MS, Peyghambarian N. Information storage and retrieval using macromolecules as storage media. In: Optical data storage. Optica Publishing Group; 2003:TuC2.
[10] Chee Yeow Meng, Kiah Han Mao, Wei Hengjia. Efficient and explicit balanced primer codes. IEEE Trans Inf Theory 2020;66(9):5344–57.
[11] Nguyen Tuan Thanh, Cai Kui, Kiah Han Mao, Dao Duc Tu, Immink Kees A Schouhamer. On the design of codes for dna computing: secondary structure avoidance codes. arXiv:2302.13714, 2023.
[12] Chu Hui, Wang Chen, Zhang Yiwei. Improved constructions of secondary structure avoidance codes for dna sequences. In: 2023 12th international symposium on topics in coding (ISTC); 2023. p. 1–5.
[13] Benerjee Krishna Gopal, Banerjee Adrish. On DNA codes with multiple constraints. IEEE Commun Lett 2021;25(2):365–8.
[14] Benerjee Krishna Gopal, Banerjee Adrish. On secondary structure avoiding dna codes with reversible and reversible-complement constraints. In: Proc. nat. conf. comm. IEEE; 2023. p. 1–6.
[15] Milenkovic Olgica, Kashyap Navin. On the design of codes for DNA computing. In: Coding and cryptography. Springer Berlin Heidelberg; 2006. p. 100–19.
[16] Wang Yixin, Noor-A-Rahim Md, Zhang Jingyun, Gunawan Erry, Guan Yong L, Poh Chueh L. Oligo design with single primer binding site for high capacity dna-based data storage. IEEE/ACM Trans Comput Biol Bioinform 2019;17(6):2176–82.
[17] Immink Kees A Schouhamer, Cai Kui. Properties and constructions of constrained codes for dna-based data storage. IEEE Access 2020;8:49523–31.