On the Design of Codes for DNA Computing: Secondary Structure Avoidance Codes

Tuan Thanh Nguyen*, Kui Cai*, Han Mao Kiah†, Duc Tu Dao†, and Kees A. Schouhamer Immink‡

* Science, Mathematics and Technology Cluster, Singapore University of Technology and Design, Singapore 487372
†School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore 637371
‡Turing Machines Inc, Willemskade 15d, 3016 DK Rotterdam, The Netherlands
Emails: {tuanthanh_nguyen, cai_kui}@sutd.edu.sg, {hmkiah,daoductu001}@ntu.edu.sg, immink@turing-machines.com

***Abstract*—In this work, we investigate a challenging problem, which has been considered to be an important criterion in designing codewords for DNA computing purposes, namely *secondary structure avoidance* in single-stranded DNA molecules. In short, secondary structure refers to the tendency of a single-stranded DNA sequence to fold back upon itself, thus becoming inactive in the computation process. While some design criteria that reduces the possibility of secondary structure formation has been proposed by Milenkovic and Kashyap (2006), the main contribution of this work is to provide an explicit construction of DNA codes that completely avoid secondary structure of arbitrary stem length.**

Formally, given codeword length n and arbitrary integer $m \geqslant 2$, we provide efficient methods to construct DNA codes of length n that avoid secondary structure of any stem length more than or equal to m. Particularly, when $m = 3$, our constructions yield a family of DNA codes of rate 1.3031 bits/nt, while the highest rate found in the prior art was 1.1609 bits/nt. In addition, for $m \geqslant 3\log n + 4$, we provide an efficient encoder that incurs only one redundant symbol.

I. Introduction

DNA computing is an emerging branch of computing that uses DNA, biochemistry, and molecular biology hardware. The field of DNA computation started with the following demonstration by Adleman in 1994 [1]. In this seminal experiment, Adleman solved an instance of the directed traveling salesperson problem by first representing each city with a synthetic DNA molecule. Then by allowing the strands to hybridize in a highly parallel fashion, Adleman obtained the desired solution. Since then, similar methods have been expanded to several attractive applications, including the development of storage technologies [2]–[5], and cell-based computation systems for cancer diagnostics and treatment [6]. Recently, the hybridization process was exploited to allow random access in DNA data storage [7].

In DNA computing, only short single-stranded DNA sequences (or *oligonucleotide sequences*) are used, where each of them is an oriented word consisting of four bases (or *nucleotides*): Adenine (**A**), Thymine (**T**), Cytosine (**C**), and Guanine (**G**). A set of encoded DNA sequences (also called DNA codewords), that satisfies certain special properties (or *constraints*) for DNA computing purposes, is called a DNA code. A broad description of the kinds of constraint problems that arise in coding for DNA computing was introduced by Milenkovic and Kashyap in 2006 [8], including *constant* **GC**-*content constraint* (refers to the percentage of nucleotides that are either **G** or **C**), *Hamming distance constraint* (that requires DNA codewords to be sufficiently different among themselves), and *secondary structure formation avoidance constraint* (that prevents DNA sequence to fold back upon itself, and consequently becoming inactive in the computation process). Similar considerations were described in [9], [10] for the design of primer address sequences in random access of DNA-based data storage systems. While constant **GC**-content constraint and Hamming distance constraint have been extensively investigated [8], [11]–[17], the study for secondary structure avoidance is much less profound.

For a DNA sequence, a secondary structure is formed by a chemically active to fold back onto itself by complementary base pair hybridization (illustrated via Figure 1). Here, the *Watson-Crick complement* is defined as: $\overline{\mathbf{A}} = \mathbf{T}, \overline{\mathbf{T}} = \mathbf{A}, \overline{\mathbf{C}} = \mathbf{G}$, and $\overline{\mathbf{G}} = \mathbf{C}$. For a sequence $\boldsymbol{x} = x_1x_2x_3\ldots x_{n-1}x_n$ over the DNA alphabet $\mathcal{D} = \{\mathbf{A}, \mathbf{T}, \mathbf{C}, \mathbf{G}\}$, the *reverse-complement* of $\boldsymbol{x}$ is defined as $\mathrm{RC}(\boldsymbol{x}) = \overline{x_n}\ \overline{x_{n-1}}\ldots\overline{x_3}\ \overline{x_2}\ \overline{x_1}$. In Figure 1, sub-sequences $\boldsymbol{x} = \mathbf{ATACC}$ and $\boldsymbol{y} = \mathrm{RC}(\boldsymbol{x}) = \mathbf{GGTAT}$ of the DNA sequence σ bind to each other after pairing of **A** with **T** and **G** with **C**, forming a secondary structure with a loop and a *stem* of length 5. DNA sequences with secondary structures are less active in the computation process [8], and hence, before reading such sequences in a wet lab, they need to be unfolded, costing more resources and energy. There exist some simple dynamic programming techniques [18], [19] that can approximately predict the secondary structures in a given DNA sequence (for example, see the Nussinov-Jacobson (NJ) algorithm in [19] as one of the most widely used schemes). Based on the NJ algorithm, the authors in [8], [13] found some design criteria that reduce the possibility of secondary structure formation in a codeword. A natural question is whether there exists efficient design of DNA constrained codes that avoid the formation of secondary structures.

It has been shown experimentally that the number of base pairs in stem regions (or *stem length*) is one important factor influencing the secondary structure of a DNA sequence. Given codeword length n and an integer $m \geqslant 2$, we study the problem of constructing DNA codes of length n that avoid secondary structure of any stem length more than or equal to m. To the best of our knowledge, this work is the first attempt aimed at providing a rigorous solution for DNA codes avoiding secondary structure for general stem lengths.

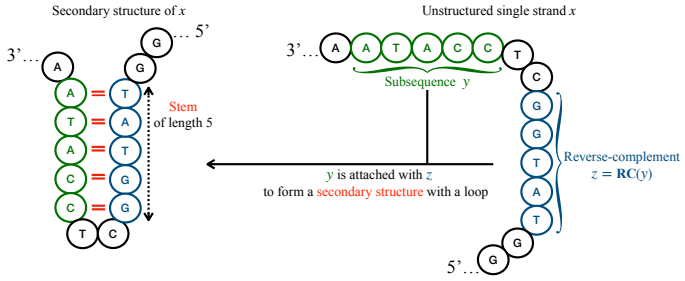


Fig. 1: DNA secondary structure model. Here, the Watson-Crick complement is: $\overline{\mathbf{A}} = \mathbf{T}, \overline{\mathbf{T}} = \mathbf{A}, \overline{\mathbf{C}} = \mathbf{G}$, and $\overline{\mathbf{G}} = \mathbf{C}$.

II. Preliminary

In this work, we use $\mathcal{D}$ to denote the DNA alphabet, where $\mathcal{D} = \{\mathbf{A}, \mathbf{T}, \mathbf{C}, \mathbf{G}\}$. Here, we have the Watson-Crick complement where $\overline{\mathbf{A}} = \mathbf{T}, \overline{\mathbf{T}} = \mathbf{A}, \overline{\mathbf{C}} = \mathbf{G}$, and $\overline{\mathbf{G}} = \mathbf{C}$.

Given two sequences $\boldsymbol{x}$ and $\boldsymbol{y}$, we let $\boldsymbol{xy}$ denote the *concatenation* of the two sequences.

Throughout this work, given a sequence $\boldsymbol{x}$ of length n, we say $\boldsymbol{y}$ is a subsequence of length k of $\boldsymbol{x}$, where $k \leqslant n$, if $\boldsymbol{y} = x_i x_{i+1} \dots x_{i+k-1}$ for some $1 \leqslant i \leqslant n-k+1$. In other words, we only consider the subsequences including consecutive symbols in $\boldsymbol{x}$. Two subsequences $\boldsymbol{y}$ and $\boldsymbol{z}$ of $\boldsymbol{x}$ are said to be *non-overlapping* if we have $\boldsymbol{y} = x_i x_{i+1} \dots x_{i+k-1}$, $\boldsymbol{z} = x_j x_{j+1} \dots x_{j+\ell-1}$, where $i > j+\ell-1$ or $j > i+k-1$.

Definition 1. For a DNA sequence $\boldsymbol{x} \in \mathcal{D}^n$, $\boldsymbol{x} = x_1 x_2 \dots x_n$, the reverse-complement of $\boldsymbol{x}$, is defined as $\mathrm{RC}(\boldsymbol{x}) = \overline{x_n}\ \overline{x_{n-1}} \dots \overline{x_3}\ \overline{x_2}\ \overline{x_1}$.

Definition 2. Given $0 < m \leqslant n$, a DNA sequence $\boldsymbol{x} \in \mathcal{D}^n$ is said to be m-secondary structure avoidance (or m-SSA) sequence if for all $k \geqslant m$, there does not exist any pair of non-overlapping subsequences $\boldsymbol{y}, \boldsymbol{z}$ of length k of $\boldsymbol{x}$ such that $\boldsymbol{y} = \mathrm{RC}(\boldsymbol{z})$. A code $\mathcal{C}$ is said to be an $(n, \mathcal{D}; m)$ SSA code if for every codeword $\boldsymbol{x} \in \mathcal{C} \cap \mathcal{D}^n$, we have $\boldsymbol{x}$ is m-SSA.

The following result is immediate.

Lemma 1. *Given $m, n > 0$, if a sequence $\boldsymbol{x} \in \mathcal{D}^n$ is m-SSA then $\boldsymbol{x}$ is m'-SSA for all $m' > m$.*

For a code $\mathcal{C} \subseteq \mathcal{D}^n$, the *code rate* is measured by the value $\log|\mathcal{C}|/n$. Intuitively, it measures the number of information bits stored in each DNA symbol. Suppose that we have an infinite family of codes $\{\mathcal{C}_n\}_{n=1}^{\infty}$, where $\mathcal{C}_n$ is a code of length n, then the asymptotic rate of the family is $\mathbf{r} \triangleq \lim_{n\to\infty} \frac{\log|\mathcal{C}_n|}{n}$. Here, we adopt the notation $\log$ to mean logarithm base two.

Definition 3. Given $m > 0$, for $n > 0$, let $\mathrm{A}(n, \mathcal{D}; m)$ be the total number of DNA sequences of length n that are m-SSA. The channel capacity, denoted by c_m, is defined by:

$$\mathrm{c}_m = \lim_{n\to\infty} \frac{\log \mathrm{A}(n, \mathcal{D}; m)}{n}.$$

The following result is immediate.

Lemma 2. *Given $m > 0$, let S_m be the set of all DNA sequences of length m such that, there is no pair of sequences $\boldsymbol{y}, \boldsymbol{z} \in S_m$, not necessary distinct, such that $\boldsymbol{y} = \mathrm{RC}(\boldsymbol{z})$. We then have $\mathrm{c}_m \leqslant 1/m \log|S_m|$.*

Observe that the size of S_m can be computed easily for constant m, a trivial upper bound is that $|S_m| \leqslant 4^m/2$, and consequently, we obtain $\mathrm{c}_2 \leqslant 1.5$ and $\mathrm{c}_3 \leqslant 1.67$.

To construct an $(n, \mathcal{D}; m)$ SSA code for arbitrary $m > 0$ by concatenation method, one can find the largest set S_N for some suitable value of N, such that, for $n = Nk$, each codeword is a concatenation of k sequences of length N from S_N and each concatenation does not create a reverse-complement subsequence from previous concatenations. The construction yields a family of DNA codes of rate $1/N \log|S_N|$ bits/nt. For example, for $m = 3$, Krishna Gopal Benerjee and Adrish Banerjee [11] constructed an $(n, \mathcal{D}; 3)$ SSA code via such a set $S = \{\mathbf{AA}, \mathbf{CC}, \mathbf{AC}, \mathbf{CA}, \mathbf{TC}\}$.

Theorem 1 (Benerjee and Banerjee [11])**.** *Set $S = \{\mathbf{AA}, \mathbf{CC}, \mathbf{AC}, \mathbf{CA}, \mathbf{TC}\}$. Let $\mathcal{C}$ be the DNA code of length $2n$ where each codeword is a concatenation of words of length two from S. We then have $\mathcal{C}$ is an $(n, \mathcal{D}; 3)$ SSA code, i.e. every codeword of $\mathcal{C}$ is 3-SSA. The size of the code is $|\mathcal{C}| = 5^n$, and the code rate is $1/2 \log 5 = 1.1609$ bits/nt.*

A. Paper Organisation and Our Main Contribution

Since the number of base pairs in stem regions (or stem length) is one important factor influencing the secondary structure of a DNA sequence, this work aims at providing a rigorous solution for $(n, \mathcal{D}; m)$ SSA codes given arbitrary m. The paper is organised as follows.

- Section III presents two efficient constructions of $(n, \mathcal{D}; m)$ SSA codes for arbitrary $m > 0$. The first construction is based on *block concatenation*, which concatenates blocks of fixed length m from a predetermined set. On the other hand, crucial to the second construction is the concept of *symbol-composition constrained codes*. Particularly, when $m = 3$, the second construction yields a family of DNA codes of rate 1.3031 bits/nt, which is higher than the code rate in [11].
- Section IV presents a linear-time encoding method for $(n, \mathcal{D}; m)$ SSA code with only one redundant symbol whenever $m \geqslant 3 \log n + 4$. The coding method is based on *sequence replacement technique*.

III. Constructions of $(n, \mathcal{D}; m)$ SSA Codes for Arbitrary $m > 0$

The first method is based on block concatenation, which concatenates blocks of length m from a predetermined set.

A. Constructions via Block Concatenation

Construction 1. *Given $m > 0$, $n = mk$ for some integer $k > 0$, set $t = \lceil m/3 \rceil$. Let S_m^* be the set of all DNA sequences of length m such that for any pair of sequences $\boldsymbol{x}_1, \boldsymbol{x}_2 \in S_m^*$, not necessary distinct, there is no pair of subsequences $\boldsymbol{y}$ of $\boldsymbol{x}_1$ and $\boldsymbol{z}$ of $\boldsymbol{x}_2$ of length t such that $\boldsymbol{y} = \mathrm{RC}(\boldsymbol{z})$. Let $\mathcal{C}$ be the DNA code of length n, where each codeword is a concatenation of k sequences of length m in S_m^*.*

Theorem 2. *The constructed code $\mathcal{C}$ from Construction 1 is an $(n, \mathcal{D}; m)$ SSA code.*

Proof. We prove the correctness of Theorem 2 by contradiction. Suppose that, there exists a codeword $\boldsymbol{c} \in \mathcal{C}, \boldsymbol{c} = \boldsymbol{x}_1\boldsymbol{x}_2 \ldots \boldsymbol{x}_k$, where $\boldsymbol{x}_i \in S_m^*$, and $\boldsymbol{c}$ is not m-SSA. In other words, there exists two non-overlapping subsequences $\boldsymbol{y}$,$\boldsymbol{z}$ of $\boldsymbol{c}$ of length $m' \geqslant m$ such that $\boldsymbol{y} = \mathrm{RC}(\boldsymbol{z})$.

Suppose that $\boldsymbol{y} = Y_1Y_2$ where Y_1 is a subsequence of $\boldsymbol{x}_i$, and Y_2 is a subsequence of $\boldsymbol{x}_{i+1}\boldsymbol{x}_{i+2} \ldots \boldsymbol{x}_{i+h}$ for some $h \geqslant 1$. We have $\boldsymbol{z} = \mathrm{RC}(Y_2)\mathrm{RC}(Y_1)$. The trivial case is if $h > 1$, or Y_2 is of length more than m, then $\boldsymbol{x}_{i+1}$ is a subsequence of Y_2 and $\mathrm{RC}(\boldsymbol{x}_{i+1})$ is a subsequence of $\boldsymbol{z}$. Clearly, if $\mathrm{RC}(\boldsymbol{x}_{i+1}) \equiv \boldsymbol{x}_j$, we have a contradiction. On the other hand, if $\mathrm{RC}(\boldsymbol{x}_{i+1}) = W_1W_2$ where W_1 is a subsequence of $\boldsymbol{x}_j$ and W_2 is a subsequence of $\boldsymbol{x}_{j+1}$ for some j, then at least one subsequence W_1 or W_2 is of size more than t, we also have a contradiction. We conclude that $h = 1$, or Y_2 is simply a subsequence of $\boldsymbol{x}_{i+1}$.

Now, since $\boldsymbol{y} = Y_1Y_2$ is of length $m' \geqslant m$, at least $Y_1 \geqslant t$ or $Y_2 \geqslant t$. W.l.o.g, assume that $Y_1 \geqslant t$.

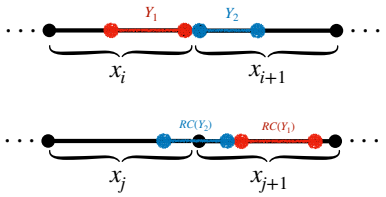
We observe that $\mathrm{RC}(Y_1)$ cannot be a subsequence of any $\boldsymbol{x}_j$ by Construction 1. In other words, $\mathrm{RC}(Y_1) = W_1W_2$ where W_1 is a subsequence of $\boldsymbol{x}_j$ and W_2 is a subsequence of $\boldsymbol{x}_{j+1}$ for some j. Similarly, we observe that the length of W_1, W_2 must be strictly smaller than t, otherwise, for example, if the length of W_1 is more than or equal to t, then two sequences $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ in S_m^* contain $\mathrm{RC}(W_1)$ and W_1 as subsequences, we have a contradiction. Since both the length of W_1, W_2 must be strictly smaller than t, causing the length of Y_1 is smaller than $2t$, we conclude that the length of Y_2 is at least t.

Now, let $\boldsymbol{U} = \mathrm{RC}(Y_2) \cap \boldsymbol{x}_{j+1}$, the subsequence that belongs to both $\boldsymbol{x}_{j+1}$ and $\mathrm{RC}(Y_2)$, which is of size at least t. We then have $\boldsymbol{U}$ is a subsequence of $\boldsymbol{x}_{j+1}$ while $\mathrm{RC}(\boldsymbol{U})$ is a subsequence of $\mathrm{RC}(\mathrm{RC}(Y_2)) = Y_2$, a subsequence of $\boldsymbol{x}_{i+1}$. We then have a contradiction.

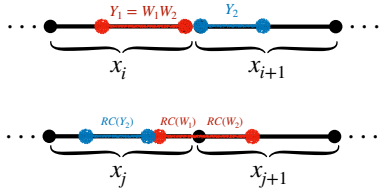
In conclusion, we have $\mathcal{C}$ is an $(n, \mathcal{D}; m)$ SSA code. We highlight our proof sketch of Theorem 2 in Figure 2. ∎

Remark 1. Observe that, the set S_m^* can be constructed via exhaustive search with complexity $O(2^m)$. In Section IV, we show that when m is sufficiently large, $m \geqslant 3\log n + 4 = \Theta(\log n)$, there exists an efficient encoding/decoding algorithm for $(n, \mathcal{D}; m)$ SSA codes with at most one redundant symbol. Hence, for the case $m = o(\log n)$, we can use Construction 1 to construct $(n, \mathcal{D}; m)$ SSA codes with complexity $2^m = \Theta(n)$.

Claim 1: When $|Y_1| \geqslant t$, we observe that $\mathrm{RC}(Y_1)$ cannot be a subsequence of any $\boldsymbol{x}_{j+1}$.



Claim 2: When $Y_1 = W_1W_2$ and $\mathrm{RC}(Y_1) = \mathrm{RC}(W_2)\mathrm{RC}(W_1)$, we must have $|W_1| \leqslant t, |W_2| \leqslant t$.



Consequently, $|Y_2| \geqslant t$, and we have $(\boldsymbol{x}_{i+1}, \boldsymbol{x}_j)$ form a pair in S_m^* that violate the condition.

Fig. 2: Proof Sketch of Theorem 2.

B. Constructions via Symbol-Composition Constrained Codes

In this subsection, we present an efficient construction for $(n, \mathcal{D}; m)$ SSA codes by simply restricting the symbol-composition for every subsequence of length m. Particularly, when $m = 3$, our method yields a family of DNA codes of rate 1.3031 bits/nt, which is higher than the code rate in [11].

High Level Description. We select a nucleotide $x \in \mathcal{D} = \{\mathbf{A}, \mathbf{T}, \mathbf{C}, \mathbf{G}\}$, and let $y = \overline{x} \in \mathcal{D}$. For some $0 < k \leqslant m$, we present an efficient method to construct an $(n, \mathcal{D}; m)$ SSA code $\mathcal{C}$ as follows. For every codeword $\boldsymbol{c} \in \mathcal{C}$, every subsequence $\boldsymbol{z}$ of length m of $\boldsymbol{c}$ contains at least k symbols x while $\boldsymbol{z}$ contains at most $(k-1)$ symbols y. We refer such a constraint to as the *symbol-composition constraint*. It is easy to verify that such a constructed code $\mathcal{C}$ is an $(n, \mathcal{D}; m)$ SSA code. Clearly, suppose on the other hand, there exists a pair of subsequences $\boldsymbol{z}_1, \boldsymbol{z}_2$ of length $\ell \geqslant m$ in $\boldsymbol{c} \in \mathcal{C}$, such that $\boldsymbol{z}_2 = \mathrm{RC}(\boldsymbol{z}_1)$. It implies that there exists two subsequences of length m, which are $\boldsymbol{z}_1'$ of $\boldsymbol{z}_1$ and $\boldsymbol{z}_2'$ of $\boldsymbol{z}_2$, and $\boldsymbol{z}_2' = \mathrm{RC}(\boldsymbol{z}_1')$. Since $\boldsymbol{z}_1'$ contains at least k symbols x, we have $\boldsymbol{z}_2' = \mathrm{RC}(\boldsymbol{z}_1')$ must contain at least k symbols $y = \overline{x}$. We then have a contradiction.

The following construction is for $m = 3$ and $k = 1$.

Construction 2 (Symbol-Composition Constrained Codes for $m = 3$, $k = 1$)**.** *Given $n > 0$, we select $x = \mathbf{A}$ and $y =$*

$\overline{x} = \mathbf{T}$. *Set* $\mathcal{D}^* = \{\mathbf{A}, \mathbf{C}, \mathbf{G}\}$. *Let* $\mathcal{C}_n$ *be the set of all DNA sequences of length* n *from alphabet* $\mathcal{D}^*$ *such that for any* $\boldsymbol{c} \in \mathcal{C}_n$*, every subsequence of length three of* $\boldsymbol{c}$ *must contain an* $\mathbf{A}$.

Theorem 3. *We have* $|\mathcal{C}_1| = 3, |\mathcal{C}_2| = 9, |\mathcal{C}_3| = 19$*, and*

$$|\mathcal{C}_n| = |\mathcal{C}_{n-1}| + 2|\mathcal{C}_{n-2}| + 4|\mathcal{C}_{n-3}|.$$

In addition, $\mathcal{C}_n$ *is an* $(n, \mathcal{D}; 3)$ *SSA code for all* $n > 0$*. The asymptotic rate of this code family is given by* $\log(\lambda) \approx 1.3031$*, where* $\lambda \approx 2.4675$ *is the largest real root of* $x^3 - x^2 - 2x - 4 = 0$.

Proof. Consider the code $\mathcal{C}_n$. For a codeword $\boldsymbol{c} \in \mathcal{C}_n$, for any subsequence $\boldsymbol{x}$ of length $\ell \geqslant 3$ of $\boldsymbol{c}$, we have $\boldsymbol{x}$ includes $\mathbf{A}$. On the other hand, since $\overline{\mathbf{A}} = \mathbf{T}$ is not used in $\boldsymbol{c}$, there is no reverse-complement of $\boldsymbol{x}$ in $\boldsymbol{c}$. In conclusion, $\boldsymbol{c}$ is 3-SSA, or $\mathcal{C}_n$ is an $(n, \mathcal{D}; 3)$ SSA code.

We now prove the cardinality of $\mathcal{C}_n$. it is easy to verify that $|\mathcal{C}_1| = 3, |\mathcal{C}_2| = 9, |\mathcal{C}_3| = 19$. For $n \geqslant 4$, we construct $\mathcal{C}_n$ recursively as follows:

$$\begin{aligned}
S_n^1 =& \{\boldsymbol{x}\mathbf{A} : \text{ for } \boldsymbol{x} \in \mathcal{C}_{n-1}\} \\
S_n^2 =& \{\boldsymbol{x}\mathbf{AC}, \boldsymbol{x}\mathbf{AG} : \text{ for } \boldsymbol{x} \in \mathcal{C}_{n-2}\} \\
S_n^3 =& \{\boldsymbol{x}\mathbf{ACC}, \boldsymbol{x}\mathbf{ACG}, \boldsymbol{x}\mathbf{AGC}, \boldsymbol{x}\mathbf{AGG} : \text{ for } \boldsymbol{x} \in \mathcal{C}_{n-3}\}, \text{and} \\
\mathcal{C}_n =& S_n^1 \cup S_n^2 \cup S_n^3.
\end{aligned}$$

In other words, S_n^1 is the set formed by concatenating all sequences in $\mathcal{C}_{n-1}$ with $\mathbf{A}$, S_n^2 is the set formed by concatenating all sequences in $\mathcal{C}_{n-2}$ with $\mathbf{AC}$ or $\mathbf{AG}$, and lastly, S_n^2 is the set formed by concatenating all sequences in $\mathcal{C}_{n-3}$ with $\mathbf{ACC}, \mathbf{ACG}, \mathbf{AGC}$, or $\mathbf{AGG}$. It is easy to verify that $S_n^i \cap S_n^j \equiv \emptyset$, and the union $S_n^1 \cup S_n^2 \cup S_n^3$ includes all possible sequences in $\mathcal{C}_n$. Therefore, we have $|\mathcal{C}_n| = |\mathcal{C}_{n-1}| + 2|\mathcal{C}_{n-2}| + 4|\mathcal{C}_{n-3}|$. ∎

Construction 2 can be generalized to construct $(n, \mathcal{D}; m)$ SSA codes with $k = 1$ as follows.

Theorem 4 (Symbol-Composition Constrained Codes for General m, $k = 1$)**.** *Given* $n, m > 0$*. Set* $\mathcal{D}^* = \{\mathbf{A}, \mathbf{C}, \mathbf{G}\}$*, and* $\mathcal{C}_n(m)$ *to be the set of all sequences* $\boldsymbol{x}$ *of length* n *from alphabet* $\mathcal{D}^*$ *such that every subsequence of length* m *of* $\boldsymbol{x}$ *include an* $\mathbf{A}$*. We then have* $|\mathcal{C}_i(m)| = 3^i$ *for* $0 \leqslant i \leqslant m-1$*, and*

$$|\mathcal{C}_n(m)| = \sum_{j=0}^{m-1} 2^j |\mathcal{C}_{n-j-1}(m)| \text{ for } n \geqslant m.$$

We then have $\mathcal{C}_n(m)$ *is an* $(n, \mathcal{D}; m)$ *SSA code for all* $n > 0$*. The asymptotic rate of this code family is given by* $\log(\lambda)$*, where* λ *is the largest real root of* $x^m - \sum_{j=0}^{m-1} 2^j x^{m-j} = 0$.

Remark 2. In general, given $m > k > 0$, set $x = \mathbf{A}$ and $y = \overline{x} = \mathbf{T}$. we use $\mathcal{C}_n(m, k)$ to denote the set of all sequences $\boldsymbol{c} \in \mathcal{D}^n$ such that every subsequence $\boldsymbol{z}$ of length m of $\boldsymbol{c}$ contains at least k symbols $\mathbf{A}$ while $\boldsymbol{z}$ contains at most $(k-1)$ symbols $\mathbf{T}$. As shown earlier, $\mathcal{C}_n(m, k)$ is an $(n, \mathcal{D}; m)$ SSA code for all m, k. A natural question is, for a given number $m > 0$, what is the value of k, where $1 \leqslant k \leqslant m$, such that the code $\mathcal{C}_n(m, k)$ has the largest cardinality? We defer the study of $\mathcal{C}_n(m, k)$, including the code's cardinality and the design of efficient encoding algorithms to map arbitrary DNA sequences into such a code, to future research work.

IV. Constructions of $(n, \mathcal{D}; m)$ SSA Codes for $m \geqslant 3\log n + 4$ with One Redundant Symbol

In this section, we show that when the stem length is sufficiently large, $m \geqslant 3\log n + 4 = \Theta(\log n)$, there exists an efficient encoding/decoding algorithm for $(n, \mathcal{D}; m)$ SSA codes with at most one redundant symbol. For simplicity, we assume that $\log_4 n$ is an integer, and define the *DNA-representation* of an integer as follows.

Definition 4. For a positive integer N, the *DNA-representation* of N is the replacement of symbols in the quaternary representation of N over $\Sigma_4 = \{0, 1, 2, 3\}$ by the following rule: $0 \leftrightarrow \mathbf{A}, 1 \leftrightarrow \mathbf{T}, 2 \leftrightarrow \mathbf{C},$ and $3 \leftrightarrow \mathbf{G}$.

Example 1. If $N = 100$, the quaternary representation of length 4 of N is 1210, hence, the DNA-representation of N is $\mathbf{TCTA}$. Similarly, when $N = 55$, the quaternary representation of length 4 of N is 0313, thus the DNA-representation of N is $\mathbf{AGTG}$.

We now present explicit construction of the encoder ENC_{SSA} and the corresponding decoder DEC_{SSA}. Our method is based on the sequence replacement technique. This method has been widely used in the literature [21]–[23]. In addition, we also restrict the length of the repeated patterns of size 2 (also known as *pattern length limited (PLL) constraint*, as introduced in [24]).

Construction of ENC_{SSA}. Given $n > m > 0$, $n > 16$, and $m \geqslant 3\log n + 4$. Set $m' = 1.5\log n + 2$. The source DNA sequence $\boldsymbol{x} \in \mathcal{D}^{n-1}$. The encoding algorithm includes three phases: *prepending phase*, *scanning and replacing phase*, and *extending phase*.

Prepending phase. The source sequence $\boldsymbol{x} \in \mathcal{D}^{n-1}$ is prepended with $\mathbf{A}$, to obtain $\boldsymbol{c} = \mathbf{A}\boldsymbol{x}$ of length n. If $\boldsymbol{c}$ is an m-SSA sequence, then the encoder outputs $\boldsymbol{c}$. Otherwise, it proceeds to the next phase.

Scanning and replacing phase. The encoder searches for the first pair of non-overlapping subsequences $\boldsymbol{y}, \boldsymbol{z}$ of length ℓ_1 of $\boldsymbol{c}$, where $\ell_1 \geqslant m'$, such that $\boldsymbol{y} = \text{RC}(\boldsymbol{z})$, or the first subsequence $\boldsymbol{u}$ of $\boldsymbol{c}$ of the form $\boldsymbol{u} = (x_1 x_2)^t$ whose length is $\ell_2 = 2t \geqslant m' = 1.5\log n + 2$, where $x_1, x_2 \in \mathcal{D} = \{\mathbf{A}, \mathbf{T}, \mathbf{C}, \mathbf{G}\}$.

- If it finds a pair of non-overlapping subsequences $\boldsymbol{y}, \boldsymbol{z}$, suppose that $\boldsymbol{c} = \boldsymbol{X}_1\boldsymbol{y}\boldsymbol{X}_2\boldsymbol{z}\boldsymbol{X}_3$, where $\boldsymbol{X}_1, \boldsymbol{X}_2, \boldsymbol{X}_3$ are subsequences of $\boldsymbol{c}$, and $\boldsymbol{y}$ starts at index i, ends at index j in $\boldsymbol{c}$, where $j = i + \ell_1 - 1$, and $\boldsymbol{z}$ starts at index k in $\boldsymbol{c}$. We have $i, j, k \leqslant n - 1$.

  Type-I Replacement. The encoder sets a pointer P_I, starting with symbol $\mathbf{T}$, and $\text{P}_\text{I} = \mathbf{T}\boldsymbol{p}_1\boldsymbol{p}_2\boldsymbol{p}_3$, where $\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3$ are the DNA-representation of $i, j,$ and k, respectively. Since $\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3$ are of length $\log_4 n$, the pointer sequence

$\mathrm{P_I}$ is of length $1+3\log_4 n = 1+1.5\log n$. It then removes $\boldsymbol{z}$ from $\boldsymbol{c}$ and prepends $\mathrm{P_I}$ to $\boldsymbol{c}$. The replacing step can be illustrated as follows.

$$\boldsymbol{X}_1\boldsymbol{y}\boldsymbol{X}_2\boldsymbol{z}\boldsymbol{X}_3 \to \boldsymbol{X}_1\boldsymbol{y}\boldsymbol{X}_2\boldsymbol{X}_3 \to \mathbf{T}\boldsymbol{p}_1\boldsymbol{p}_2\boldsymbol{p}_3\boldsymbol{X}_1\boldsymbol{y}\boldsymbol{X}_2\boldsymbol{X}_3$$

Noted that the removed sequence $\boldsymbol{z}$ is of length $\ell_1 \geqslant m' = 1.5\log n + 2$, while the insertion pointer P_I is of length $1.5\log n + 1$. Consequently, such a replacement reduces the length of the current sequence by at least one symbol.

- On the other hand, suppose that it finds a subsequence $\boldsymbol{u}$ of $\boldsymbol{c}$ of the form $\boldsymbol{u} = (x_1x_2)^t$ whose length is $\ell_2 = 2t \geqslant m'$, where $x_1, x_2 \in \mathcal{D} = \{\mathbf{A}, \mathbf{T}, \mathbf{C}, \mathbf{G}\}$. We further suppose that $\boldsymbol{c} = \boldsymbol{U}_1(x_1x_2)^t\boldsymbol{U}_2$, where $\boldsymbol{U}_1, \boldsymbol{U}_2$ are subsequences of $\boldsymbol{c}$, and $\boldsymbol{u}$ starts at index i, and ends at index j in $\boldsymbol{c}$, where $j = i + \ell_2 - 1$. We have $i, j \leqslant n-1$.

  Type-II Replacement. Similarly, the encoder sets a pointer $\mathrm{P_{II}}$, starting with symbol $\mathbf{C}$, and $\mathrm{P_{II}} = \mathbf{C}x_1x_2\boldsymbol{q}_1\boldsymbol{q}_2$, where $\boldsymbol{q}_1, \boldsymbol{q}_2$ are the DNA-representation of i and j, respectively. Since $\boldsymbol{q}_1, \boldsymbol{q}_2$ are of length $\log_4 n$, the pointer sequence $\mathrm{P_{II}}$ is of length $1 + 2 + 2\log_4 n = 3 + \log n$. It then removes $(x_1x_2)^{\ell_2}$ from $\boldsymbol{c}$ and prepends $\mathrm{P_{II}}$ to $\boldsymbol{c}$. The replacing step can be illustrated as follows.

$$\boldsymbol{U}_1(x_1x_2)^t\boldsymbol{U}_2 \to \boldsymbol{U}_1\boldsymbol{U}_2 \to \mathbf{C}x_1x_2\boldsymbol{q}_1\boldsymbol{q}_2\boldsymbol{U}_1\boldsymbol{U}_2.$$

  Noted that the removed sequence is of length $\ell_2 \geqslant m' = 1.5\log n + 2$, while the insertion pointer $\mathrm{P_{II}}$ is of length $\log n + 3$. Hence, such a replacement reduces the length of the current sequence by at least $(0.5\log n - 1)$ symbols. Observe that $0.5\log n - 1 > 1$ for $n > 16$.

The encoder repeats the scanning and replacing steps until the current sequence $\boldsymbol{c}$ contains no pair of non-overlapping subsequences of length more than or equal to m' such that one is the reverse-complement of the other, no subsequence $\boldsymbol{u}$ of the form $\boldsymbol{u} = (x_1x_2)^t$ whose length is $\ell_2 = 2t \geqslant m'$, or the current sequence is of length $m' - 1$. Note that each replacement (either Type-I or Type-II) reduces the length of the current sequence by at least one symbol, and hence, this procedure is guaranteed to terminate. Here, we also note that the order of the scanning step is defined according to the starting index of the corresponding subsequences. In case the first subsequence $\boldsymbol{y}$ forming a secondary structure, is also the starting of such a subsequence $\boldsymbol{u}$, the encoder proceeds to type-I replacement.

Extending phase. If the length of the current sequence $\boldsymbol{c}$ is N_0 where $N_0 < n$, the encoder appends a suffix of length $N_1 = n - N_0$ to obtain a sequence of length n. Surprisingly, regardless the choice of the appending suffix, there is an efficient algorithm to decode the source DNA sequence uniquely (refer to the construction of $\textsc{Dec}_{\mathrm{SSA}}$). Here, we present one efficient method to generate a suitable suffix so that the output codeword remains m-SSA.

- If N_1 is even, we append $\boldsymbol{s} = (\mathbf{AC})^{N_1/2}$ to the end of $\boldsymbol{c}$.
- If N_1 is odd, we append $\boldsymbol{s} = (\mathbf{AC})^{(N_1-1)/2}\mathbf{A}$ to the end of $\boldsymbol{c}$.

Theorem 5. *The encoder* $\textsc{Enc}_{\mathrm{SSA}}$ *is correct. In other words,* $\textsc{Enc}_{\mathrm{SSA}}(\boldsymbol{x})$ *is an* m*-SSA sequence of length* n *for all* $\boldsymbol{x} \in \mathcal{D}^{n-1}$. *The redundancy of* $\textsc{Enc}_{\mathrm{SSA}}$ *is one redundant symbol.*

Proof. Suppose that $\boldsymbol{c} = \textsc{Enc}_{\mathrm{SSA}}(\boldsymbol{x}) \in \mathcal{D}^n$, and $\boldsymbol{c} = \boldsymbol{c}_1\boldsymbol{s}$, where $\boldsymbol{c}_1$ is m'-SSA and the length of the repeated patterns of size 2 in $\boldsymbol{c}_1$ is of length at most $m' = 1.5\log n + 2$, and $\boldsymbol{s}$ is the generated suffix of $\boldsymbol{c}_1$ at the extending phase. Consider an arbitrary sequence $\boldsymbol{y}$ of length $\ell \geqslant 3\log n + 4$. Suppose that $\boldsymbol{y} = \boldsymbol{y}_1\boldsymbol{y}_2$, where $\boldsymbol{y}_1$ is a subsequence of $\boldsymbol{c}_1$ while $\boldsymbol{y}_2$ is a subsequence of $\boldsymbol{s}$. We have the following cases.

- If $\boldsymbol{y}_1$ is of length less than m' (particularly including the case $\boldsymbol{y}_1 \equiv \varnothing$), hence the length of $\boldsymbol{y}_2$ is more than m'. Clearly, there is no subsequence $\boldsymbol{z}$ in $\boldsymbol{c}_1\boldsymbol{s}$ that $\boldsymbol{y} = \mathrm{RC}(\boldsymbol{z})$, as the length of the repeated patterns of size 2 in $\boldsymbol{c}_1$ is of length at most m'.
- If $\boldsymbol{y}_1$ is of length more than or equal to m', we also conclude that there is no subsequence $\boldsymbol{z}$ in $\boldsymbol{c} = \boldsymbol{c}_1\boldsymbol{z}$ that $\boldsymbol{y} = \mathrm{RC}(\boldsymbol{z})$ since $\boldsymbol{c}_1$ is m'-SSA. ■

We now present the corresponding decoding algorithm.

Construction of $\textsc{Dec}_{\mathrm{SSA}}$. From a DNA sequence $\boldsymbol{c}$ of length n, the decoder scans from left to right. If the first symbol is $\mathbf{A}$, the decoder simply removes $\mathbf{A}$ and identifies the last $(n-1)$ symbols as the source sequence. On the other hand,

- if it starts with $\mathbf{T}$, the decoder takes the prefix of length $(1 + 1.5\log n)$ and concludes that this prefix is a pointer prepended after a type-I replacement. In other words, the pointer is of the form $\mathbf{T}\boldsymbol{p}_1\boldsymbol{p}_2\boldsymbol{p}_3$, where $\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3$, each is of length $\log_4 n = 0.5\log n$. The decoder sets i, j, k to be the positive integers whose DNA-representations are $\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{p}_3$, respectively and sets $\boldsymbol{y}$ to be the subsequence containing the symbols from index i to index j. It removes the pointer, adds $\boldsymbol{z} \equiv \mathrm{RC}(\boldsymbol{y})$ to $\boldsymbol{c}$ at index k.
- if it starts with $\mathbf{C}$, the decoder takes the prefix of length $(3 + \log n)$ and concludes that this prefix is a pointer prepended after a type-II replacement. In other words, the pointer is of the form $\mathbf{C}x_1x_2\boldsymbol{q}_1\boldsymbol{q}_2$, where $\boldsymbol{q}_1, \boldsymbol{q}_2$, each is of length $\log_4 n = 0.5\log n$. The decoder sets i, j to be the positive integers whose DNA-representations are $\boldsymbol{q}_1, \boldsymbol{q}_2$, respectively. It then removes the pointer, adds $\boldsymbol{z} \equiv (x_1x_2)^{(j-i+1)/2}$ to $\boldsymbol{c}$ at index i.

The decoding procedure terminates when the first symbol is $\mathbf{A}$, and takes the following $(n-1)$ symbols as the user data.

Complexity analysis. For codeword of length n, the time complexity of the encoder (and the corresponding decoder) is linear in n, which follows from: the number of replacing operations is at most $n-m$, which is $\Theta(n)$, and the complexity of the each replacing operation (including the prepending prefix step or converting quaternary representation to DNA-representation of an integer) is constant time $\Theta(1)$.

V. Conclusion

We have presented efficient algorithms to construct DNA codes that avoid secondary structure of arbitrary stem length. Particularly, when $m \geqslant 3\log n + 4$, we have provided an efficient encoder that incurs only one redundant symbol, and when $m = 3$, our constructions yield a family of DNA codes of rate 1.3031 bits/nt, that improve the previous highest code rate in the literature.

References

[1] L. M. Adleman, "Molecular computation of solutions to combinatorial problems," *Science*, vol. 266, pp. 1021-1024, Nov. 1994.

[2] G. M. Church, Y. Gao, and S. Kosuri, "Next-generation digital information storage in DNA," *Science*, vol. 337, no. 6102, pp. 1628-1628, 2012.

[3] Y. Erlich and D. Zielinski, "DNA fountain enables a robust and efficient storage architecture," *Science*, vol. 355, no. 6328, pp. 950-954, 2017.

[4] L. Organick, S. Ang, Y. J. Chen, R. Lopez, S. Yekhanin, K. Makarychev, M. Racz, G. Kamath, P. Gopalan, B. Nguyen, C. Takahashi, S. Newman, H. Y. Parker, C. Rashtchian, K. Stewart, G. Gupta, R. Carlson, J. Mulligan, D. Carmean, G. Seelig, L. Ceze, and K. Strauss, "Random access in large-scale DNA data storage", *Nature Biotechnology*, vol. 36, 242–248, 2018.

[5] N. Goldman, P. Bertone, S. Chen, C. Dessimoz, E. M. LeProust, B. Sipos, and E. Birney, "Towards practical, high-capacity, low-maintenance information storage in synthesized DNA," *Nature*, vol. 494, 77-80, 2013.

[6] Y. Benenson, B. Gil, U. Ben-Dor, R. Adar and E. Shapiro, "An autonomous molecular computer for logical control of gene expression," *Nature*, vol. 429, pp. 423-429, May 2004.

[7] S. M. H. T. Yazdi, S. M., R. Gabrys and O. Milenkovic, "Portable and error-free DNA-based data storage," *Scientific reports*, 7(1), 1-6, 2017.

[8] O. Milenkovic and N. Kashyap, "On the design of codes for DNA computing," in *Coding Cryptogr.*, Germany: Springer, Mar. 2006, pp. 100-119.

[9] S. M. H. T. Yazdi, H. M. Kiah, H. M., R. Gabrys, and O. Milenkovic, "Mutually uncorrelated primers for DNA-based data storage," *IEEE Transactions on Information Theory*, 64(9), 6283-6296, 2018.

[10] Y. M. Chee, H. M. Kiah and H. Wei, "Efficient and explicit balanced primer codes," *IEEE Transactions on Information Theory*, 66(9), 5344–5357, 2020.

[11] K. G. Benerjee and A. Banerjee, "On DNA Codes With Multiple Constraints," in *IEEE Communications Letters*, vol. 25, no. 2, pp. 365-368, Feb. 2021, doi: 10.1109/LCOMM.2020.3029071.

[12] K. G. Benerjee, S. Deb, and M. K. Gupta, "On conflict free DNA codes", *Cryptogr. Commun.* **13**, 143-171, 2021. https://doi.org/10.1007/s12095-020-00459-7.

[13] O. Milenkovic and N. Kashyap, "DNA codes that avoid secondary structures," in *Proceedings. International Symposium on Information Theory*, Sep. 2005, pp. 288-292.

[14] T. T. Nguyen, K. Cai, K. A. Schouhamer Immink and H. M. Kiah, "Capacity-Approaching Constrained Codes With Error Correction for DNA-Based Data Storage," in *IEEE Transactions on Information Theory*, vol. 67, no. 8, pp. 5602-5613, Aug. 2021, doi: 10.1109/TIT.2021.3066430.

[15] O.D. King, "Bounds for DNA codes with constant GC-content," *The Electronic Journal of Combinatorics*, vol. 10, no. 1, R33, 2003.

[16] P. Gaborit and O.D. King, "Linear constructions for DNA codes," *Theoretical Computer Science*, vol. 334, no. 1-3, pp. 99-113, April 2005.

[17] K. Cai, H. M. Kiah, M. Motani and T. T. Nguyen, "Coding for Segmented Edits with Local Weight Constraints," *2021 IEEE International Symposium on Information Theory (ISIT)*, 2021, pp. 1694-1699, doi: 10.1109/ISIT45174.2021.9517851.

[18] K. Breslauer, R. Frank, H. Blocker, and L. Marky, "Predicting DNA duplex stability from the base sequence," *Proc. Natl. Acad. Sci.* USA, vol. 83, pp. 3746-3750, 1986.

[19] R. Nussinov and A.B. Jacobson, "Fast algorithms for predicting the secondary structure of single stranded RNA ," *Proc. Natl. Acad. Sci.*, USA, vol. 77, no. 11, pp. 6309-6313, 1980.

[20] R. Heckel, G. Mikutis, and R. N. Grass, "A characterization of the DNA data storage channel," *Sci. Rep.*, vol. 9, no. 1, pp. 1-12, Jul. 2019.

[21] T. Thanh Nguyen, K. Cai and K. A. Schouhamer Immink, "Binary Subblock Energy-Constrained Codes: Knuth's Balancing and Sequence Replacement Techniques," *2020 IEEE International Symposium on Information Theory (ISIT)*, Los Angeles, CA, USA, 2020, pp. 37-41, doi: 10.1109/ISIT44484.2020.9174430.

[22] O. Elishco, R. Gabrys, M. Medard, and E. Yaakobi, "Repeated-Free Codes", *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Paris, France, 2019.

[23] T. T. Nguyen, K. Cai, H. M. Kiah, K. A. Schouhamer Immink and Y. M. Chee, "Using One Redundant Bit to Construct Two-Dimensional Almost-Balanced Codes," *2022 IEEE International Symposium on Information Theory (ISIT)*, 2022, pp. 3091-3096, doi: 10.1109/ISIT50566.2022.9834724.

[24] S. Wang, J. Sima and F. Farnoud, "Non-binary Codes for Correcting a Burst of at Most 2 Deletions," *2021 IEEE International Symposium on Information Theory (ISIT)*, Melbourne, Australia, 2021, pp. 2804-2809, doi: 10.1109/ISIT45174.2021.9517917.