Codecs for DNA-based Data Storage Systems with Multiple Constraints for Internet of Things

Kaixin Fan*, Huaming Wu* and Ruidong Li†
*Center for Applied Mathematics, Tianjin University, Tianjin 300072, China
†Institute of Science and Engineering, Kanazawa University, Kanazawa 920-1192, Japan
E-mail: {kxfan, whming}@tju.edu.cn, liruidong@ieee.org

GLOBECOM 2023 - 2023 IEEE Global Communications Conference | 979-8-3503-1090-0/23/$31.00 ©2023 IEEE | DOI: 10.1109/GLOBECOM54140.2023.10437031

***Abstract*—Internet of Things (IoT) devices are severely constrained in computational capacity, battery life, and data storage, which fail to meet the requirement of mass data storage. With the explosive growth of data to be stored, Deoxyribonucleic acid (DNA)-based storage has become a promising direction for IoT data storage due to its various advantages, e.g. high capacity, long durability and scalability. However, DNA synthesis and sequencing are subject to errors due to certain biochemical properties of DNA. In this paper, an explicit encoding and decoding scheme for constrained systems satisfying both 3-RLL constraint and strong-(4,1)-locally-GC-balanced constraint is designed. We propose the use of a state-splitting algorithm to encode binary strong-(4,1)-locally-balanced constrained systems with the rate $2:3$, and a state-dependent decoding algorithm to decode the encoded data. The calculation results show that the codebook of the encoding scheme in this paper is larger than that of the existing scheme, and the total number of codewords with a length of 24 is more than 6 times that of the existing scheme. The information rate is higher than that of existing coding schemes. The encoding table size required is two orders of magnitude smaller than the existing scheme.**

***Index Terms*—DNA-based storage, Constrained systems, Encode and decode, Internet of Things.**

I. Introduction

With the rapid development of the Internet of Things (IoT) and the next generation of information technology, massive data are collected every day by different types of sensors, however, the processing, transmission and storage of big data bring new challenges to mobile users. As far as we know, wireless sensor networks are an essential part of IoT. However, due to the limited size, IoT devices are still severely constrained in computational capacity, battery life, and data storage. Moreover, the exponential growth of data globally has presented challenges for traditional data storage methods, particularly in terms of preserving and protecting data.

As a data carrier, Deoxyribonucleic acid (DNA) molecules boast several advantages compared to traditional storage media, including high storage density, long lifespan, and low maintenance costs. These characteristics make DNA a promising alternative for data storage and it is envisioned to have a wide range of potential applications in the future. Therefore, DNA-based storage shows a strong correlation and promising direction for IoT communication technology. However, the researchers found that during the process of DNA storage in vitro, especially during DNA synthesis, Polymerase Chain Reaction (PCR), and DNA sequencing, base substitution, deletion, and insertion errors are common when DNA strands do not meet biochemical constraints. In particular, the DNA strand should meet the k-RLL constraint (i.e. the repetition of the same homopolymer cannot exceed k.) and the strong-(l,δ)-locally-GC-balanced constraint (i.e. in windows of length greater than or equal to l, the proportion of guanine(G) and cytosine(C) is within a certain range).

A DNA storage system that meets these constraints is referred to as a constrained system (S). Ross *et al.* [1] experimentally found that substitution and deletion errors increased significantly when the repeats of the same nucleotide on the nucleotide chain exceeded 6. In most synthesis and sequencing techniques during DNA storage, too high or low GC-content in the DNA strand can significantly increase error occurrence [2]. The global GC-content constraint refers to the requirement that the GC-content in the DNA strand must fall within a specified range. Recently, the global GC-content constraint has been widely studied in the literature [3]–[7]. However, the local GC-content constraint has been shown to be more appropriate for DNA storage [8] compared to global GC-content constraint, due to its ability to better predict the success of polymerase chain reaction (PCR). This is due to the fact that local GC-content constraint provides a better predictor of polymerase chain reaction (PCR) success and is more sensitive in forecasting PCR results.

On the local GC-content constraint, Gabrys *et al.* [9] put forward the concept of strong-(l,δ)-locally-balanced constraint in binary and provided its capacity, but no codec scheme was presented. Wang *et al.* [10] proposed two methods for encoding the strong-(l,δ)-locally-balanced constraint system. The first one is constructing two one-to-one correspondence tables of blocks of length s to the Dyck path. However, this method can become impractical when the parameters are large as it requires a significant amount of entries to be stored, making it difficult to encode and decode. The second method uses state transition diagrams and the running digital sum (RDS) sequence to encode the strong-(4,1)-locally-balanced constrained code. However, the establishment of the state transition graphs lacks a theoretical basis. Fan *et al.* [11] calculated the capacity of constrained channels that satisfy both k-RLL and strong-(l,δ)-locally-GC-balanced constraints.

In this paper, we present an explicit encoding and decoding scheme for constrained systems that satisfy both the 3-RLL constraint and the strong-(4,1)-locally-GC-balanced constraint. In this coding scheme, the constraint graph is split by a state-splitting algorithm and decoded by a state-dependent decoding algorithm. We describe the theoretical basis of the encoding and decoding scheme, give the encoder of the binary sequence, and analyze the performance of the encoder through calculation. The main contribution of this work is twofold:

- We simplify the coding problem of a quaternary constrain system, which satisfies both the 3-RLL and the strong-(4,1)-locally-GC-balanced constraint to the coding problem of a binary constrained system which satisfies the strong-(4,1)-locally-balanced constraint. Therefore, coding and decoding can be designed in binary-constrained systems, which makes the problem simple and easy.
- We use the state-splitting algorithm to obtain the encoder $\boldsymbol{\epsilon}$ for a binary-constrained system with encoding rate 2:3 that satisfies the strong-(4,1)-locally-balanced constraint. This encoder can encode any binary sequence of length $2n$ into a binary sequence of length $3n$ that satisfies the strong-(4,1)-locally-balanced constraint. Compared with the existing scheme, it is found that the codebook of $\boldsymbol{\epsilon}$ is larger than the codebook of the existing scheme, and the total number of codewords with a length of 24 is more than 6 times that of the existing scheme.

II. Definitions and Preliminaries

For a binary word $\boldsymbol{x} \in \Sigma_q^n = \{0, 1, \ldots, q-1\}^n$, the function $wt(\boldsymbol{x})$ represents the weight of $\boldsymbol{x}$. When $q = 2$, the weight refers to the proportion of "1"; When q is equal to 4, it's the ratio of 2(C) and 3(G).

Definition 1. **q-ary k-RLL constraint***: Let $\boldsymbol{x} = (x_1, x_2, \ldots, x_n) \in \Sigma_q^n = \{0, 1, \ldots, q-1\}^n$. Given $k > 0$, we say that $\boldsymbol{x}$ is k-RLL constrained if the run of consecutive symbol of Σ_q is at most k.*

Definition 2. **q-ary strong-(l,δ)-locally-(GC-)balanced constraint***: Let l be an even positive integer and δ be a nonnegative integer. A sequence $\boldsymbol{x} \in \Sigma_q^n$ is said to satisfy the q-ary strong-(l,δ)-locally-(GC-)balanced constraint if for all even l', $l' \geq l$ and $1 \leq i \leq n - l' + 1$, the weight of the window $\boldsymbol{x}[i; l'] = (x_i, x_{i+1}, \ldots, x_{i+l'-1})$ is between $\left[\frac{l'}{2} - \delta, \frac{l'}{2} + \delta\right]$, i.e. $\frac{l'}{2} - \delta \leq w(\boldsymbol{x}[i; l']) \leq \frac{l'}{2} + \delta$.*

We adopt a quaternary model of DNA data storage coding table based on the base characteristics of DNA, as shown in Table I. And the binary sequence $\boldsymbol{x} \in \Sigma_2^{2n}$ can be divided into odd sequence $\boldsymbol{x_o} = (x_1, x_3, \ldots, x_{2n-1})$ and even sequence $\boldsymbol{x_e} = (x_2, x_4, \ldots, x_{2n})$.

TABLE I: A Quaternary Model of DNA Data Storage Coding

binary data	00	01	10	11
base	A	T	C	G
quaternary data	0	1	2	3

Definition 3. **Deterministic***: The labeled graph G is considered deterministic if the outgoing edges from each state are labeled uniquely [12].*

Definition 4. **Finite Local Anticipation (FLA)***: A finite-state transition graph is said to have finite local anticipation a if a is the smallest nonnegative integer such that for each state u, all paths starting from u and producing sequences of the same length $a + 1$ have the same initial edge label.*

III. Code Constructions

In this section, we focus on developing an efficient code construction method to encode arbitrarily long binary source data into a sequence satisfying the 3-RLL constraint and strong-(l,δ)-locally-GC-balanced constraint. In this section, we first simplify the channel and then use the state-splitting algorithm to design an encoder.

According to proposition 1 in [11], when $k \geq \frac{l}{2} + \delta$, by using Table I, we can combine the binary odd sequence of length n satisfying both binary k-RLL constraint and binary strong-(l,δ)-locally-balanced constraint with any binary even sequence of length n to obtain the quaternary sequence of length n satisfying both k-RLL constraint and strong-(l, δ)-locally-GC-balanced constraint. Since $k = 3, l = 4, \delta = 1$ satisfies the conditions, the codec problem of a quadrilateral constrained system satisfying both 3-RLL constraint and strong-(4,1)-locally-GC-balanced constraint is transformed into a binary constrained system codec problem satisfying both binary 3-RLL constraint and strong-(4,1)-locally-balanced constraint. By the definition of 3-RLL constraint and strong-(4,1)-locally-balanced constraint, it can be concluded that strong-(4,1)-locally-balanced constraint includes 3-RLL constraint. Therefore, the binary-constrained system codec problem satisfying both the binary 3-RLL constraint and the strong-(4,1)-locally-balanced constraint can be simplified to the binary-constrained system codec problem satisfying the strong-(4,1)-locally-balanced constraint. Next, we design an encoder for a binary-constrained system that satisfies the strong-(4,1)-locally-balanced constraint.

Theorem 1. *(***Finite-State Coding Theorem***) Given a constrained system S with capacity $\mathbb{C}(S)$ and positive integers p and q such that the inequality $p/q \leq \mathbb{C}(S)$ holds, there exists a finite-state encoder with a state-dependent decoder that encodes an arbitrary binary sequence into a constrained system S at a constant rate $p : q$ [13].*

Theorem 1 states that for any given positive integers p and q such that $p/q \leq \mathbb{C}(S)$, a code that operates at rate $p : q$ exists. Specifically, if p and q are selected as relatively prime numbers, it is possible to design an encoder with a codeword length that is minimum and consistent with the chosen rate $R = p : q$. Finite-state graphs are a widely utilized method for the representation of constrained sequences [10], [14]. In our work, we adopt the state-splitting algorithm [13], [15] to design a binary constrained system encoder with an encoding rate of $2 : 3$ which satisfies the strong-(4,1)-locally-balanced constraint. It can encode any binary sequence with a length of $2n$ into a binary sequence with a length of $3n$ which satisfies the strong-(4,1)-locally-balanced constraint.

A. State-Splitting Algorithm

In our work, the rate of the encoder is mathematically represented as $R = p/q$. A finite-state encoder, with this specified rate, is defined as a labeled encoder that operates by accepting a binary p-block as its input sequence and producing a binary q-block as its corresponding output label. The symbol $\boldsymbol{\epsilon}$ is employed to denote such a labeled finite-state encoder. The design of this type of block coding is comprised of several essential steps, which are outlined as follows.

- **Step 1**: we must identify a deterministic labeled graph G that represents the given constrained system S. This is often straightforward as most constrained systems have a natural deterministic representation.
- **Step 2**: we compute the adjacency matrix A_G of G. Using this matrix, we can calculate the capacity of the constrained system through $\mathbb{C}(S) = \log \lambda(A_G)$, where $\lambda(A_G)$ denotes the largest characteristic real root of matrix A_G. Subsequently, we select a code rate $R = p : q$ that satisfies the inequality
$$\frac{p}{q} \leq \mathbb{C}(S). \tag{1}$$
- **Step 3**: Construct G^q.
- **Step 4**: Find an $(A_G^q, 2^p)$-approximate eigenvector $\boldsymbol{v}$ using the Franaszek algorithm [16]. Eliminate all the states in G^q with components equal to 0 and restrict it to an irreducible component H. Repeat steps 5-6 until a labeled graph H with the minimum out-degree of at least 2^p per state is obtained.
- **Step 5**: Find a $\boldsymbol{v}$-consistent partition of the state in H.
- **Step 6**: Find the $\boldsymbol{v}$-consistent splitting corresponding to this partition and create a new labeled graph H' along with a new approximate eigenvector $\boldsymbol{v'}$.
- **Step 7**: Delete the excess edges from each state in H' so that each state has exactly 2^p output edges. The remaining 2^p edges are then tagged with p-blocks for each edge, resulting in each edge having both a source word and a codeword. This process results in obtaining a finite-state encoder with rate $p : q$ for the constrained system S.

It should be noted that the intelligent selection of the $\boldsymbol{v}$-consistent partition of the state in the state-splitting algorithm and the intelligent allocation of the p-block input label on the output edge of the 2^p bar can reduce the complexity of the encoder and decoder. It can be seen from the above steps that this algorithm is applicable to all constrained systems with deterministic representations.

B. Finite-State Encoder for Strong-(4,1)-Locally-Balanced Constrained System S_0

Following the above steps, we find a deterministic representation of S_0, which is depicted in Fig. 1 as the constraint graph G.

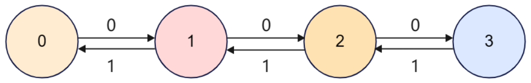

Fig. 1: The deterministic constrained graph

According to the Perron-Frobenius theorem [17], the capacity of S_0 can be determined. The adjacency matrix of the constrained graph G is: $A_G = \begin{pmatrix} 0&1&0&0\\ 1&0&1&0\\ 0&1&0&1\\ 0&0&1&0 \end{pmatrix}$. After calculation, the maximum eigenvalue is determined to be $\lambda_{max} = \frac{\sqrt{5}+1}{2}$. Therefore, the capacity can be derived as follows: $\mathbb{C}(S_0) = \log_2(\lambda_{\max}) = \log_2(\frac{\sqrt{5}+1}{2}) = 0.6942$.

When selecting values for p and q, it is common to prioritize the minimization of both p and q while still satisfying the capacity limit given in (1), as this helps to reduce the complexity of the system. Consider the case where $p = 2$ and $q = 3$ are chosen and meet the conditions of (1). By following the steps outlined above, the graph $G^q = G^3$ can be constructed. The adjacency matrix of G^3 is given as: $A_G^3 = \begin{pmatrix} 0&2&0&1\\ 2&0&3&0\\ 0&3&0&2\\ 1&0&2&0 \end{pmatrix}$. The graph G^3 is illustrated in Fig. 2.

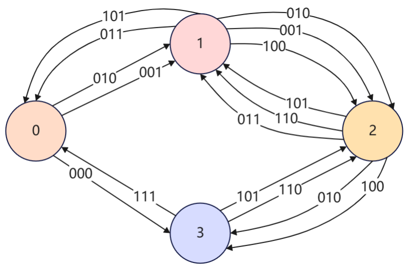

Fig. 2: G^3

The condition that allows us to utilize G^q as a finite-state encoder with rate $p : q$ is that it has at least four outgoing edges per state. This condition can be expressed in terms of the adjacency matrix of G^q as the approximate eigenvector inequality: $A^q\boldsymbol{v} \geq 2^p\boldsymbol{v}$, where the elements of vector $\boldsymbol{v}$ are non-negative integers, and $\boldsymbol{v}$ is referred to as the $(A_G^q, 2^p)$-approximate eigenvector. The Franaszek algorithm [16] is then employed to find an $(A_G^3, 2^2)$-approximate eigenvector $\boldsymbol{v}$. The algorithmic flow is described in Algorithm 1.

Algorithm 1 Franaszek algorithm

```
Require: $\xi \in \mathbb{N}^+$
Ensure: $\boldsymbol{v}$
1: $\boldsymbol{y} \leftarrow \boldsymbol{\xi}$;
2: $\boldsymbol{v} \leftarrow \boldsymbol{0}$;
3: while $\boldsymbol{v} \neq \boldsymbol{y}$ do
4:     $\boldsymbol{v} \leftarrow \boldsymbol{y}$ ;
5:     $\boldsymbol{y} \leftarrow \min\{\lfloor \frac{1}{n} A\boldsymbol{v} \rfloor, \boldsymbol{v}\}$ ;
6: return $\boldsymbol{v}$
```

By using this method, we get that the $(A_G^3, 2^2)$-approximate eigenvector is $\boldsymbol{v} = (1, 2, 2, 1)$ when $\xi = 2$. Since every term of $\boldsymbol{v}$ is non-zero, states are not deleted, and $H = G^3$. This approximate eigenvector is utilized to guide the state-splitting process. The vector $\boldsymbol{v}$ acts as allocation weights, where v_i

represents the weight assigned to state i. As can be seen from vector $\boldsymbol{v}$, states 0 and 3 do not undergo splitting, while states 1 and 2 are split into two states, respectively. These subsequent states are labeled as $1^{(1)}, 1^{(2)}$ and $2^{(1)}, 2^{(2)}$.

$\boldsymbol{v}$-consistent partition of the state: Let E_u represent the set of outgoing edges from state u. Each outgoing edge is assigned a weight based on the weight assigned to its terminal state. For instance, the weight of the outgoing edge "101" from state 1 is 1, which corresponds to its terminal state 0. According to the weights corresponding to the approximate eigenvectors $\boldsymbol{v}$, we need to split E_u into v_i disjoint sets, $\{E_u^1, \ldots, E_u^{v_i}\}$. The partitioning is performed based on the following criteria: $\sum_{j\in E_u^1} v_j \bmod 2^2 = 0, v_i - 2^{-2}\sum_{j\in E_u^1} v_j \leq 0$. The first condition means that the sum of the weights of the edges in each set of edges must be a multiple of 4. The second condition dictates that the sum of the weights of the edges must not exceed 4 times the weight of the state, as designated by the vector $\boldsymbol{v}$. In the case of state 1, $E_1 = \{101, 011, 010, 001, 100\}$, with a weight of 2 and 5 outgoing edges, there are 15 possible ways to partition the outgoing edges, yet only three of them satisfy the aforementioned conditions. For demonstration purposes, we arbitrarily choose one of these three valid partitions, such as $E_1^1 = \{101, 011, 010\}$ and $E_1^2 = \{001, 100\}$. In a similar manner, the outgoing edges of state 2 are partitioned into $E_2^1 = \{101, 110\}$ and $E_2^2 = \{011, 010, 100\}$. According to the established edge-change rule, a new graph H' will be generated after the splitting of states 1 and 2.

The edges in graph H' that are not associated with state splitting are inherited from graph G^3. In the event that state u is partitioned into two states, $u^{(1)}$ and $u^{(2)}$, we will examine the two distinct scenarios regarding the edges involved in the state splitting.

- **Case 1**: Consider an edge e in the graph G^3 that starts at state $u' \neq u$ and ends at state u. This edge is duplicated in G^3 and creates two new edges: $e^{(1)}$ from u' to $u^{(1)}$ and $e^{(2)}$ from u' to $u^{(2)}$.
- **Case 2**: Consider an edge e in the graph G^3 that starts at state u and ends at state u'. Assuming that edge e is part of set E_u^i in the partition E_u, the graph H' will have a corresponding edge from state $u^{(i)}$ to state u'.

Additionally, the labels assigned to the edges are derived directly from the labels present in G^3. The resulting graph, represented by H', is visualized in Fig. 3.

Each state in H' has at least 2^2 outgoing edges. To satisfy the requirement of having only 4 outgoing edges per state, the excess edges are randomly removed. For instance, the edge labeled "000" is removed from state 0, and the edge labeled "110" is removed from state 3. The 2-block source word is assigned to the four remaining outgoing edges as input tags, resulting in a finite-state encoder $\boldsymbol{\epsilon}$ with a rate 2:3 for S_0, as depicted in Fig. 4.

We will use the following example to demonstrate the coding process. The encoder returns a status sequence when generating a code word sequence.

Example: Assume that the input binary source code is: $\boldsymbol{x_1} = 011110001111$ (In the decoding example, it is shown

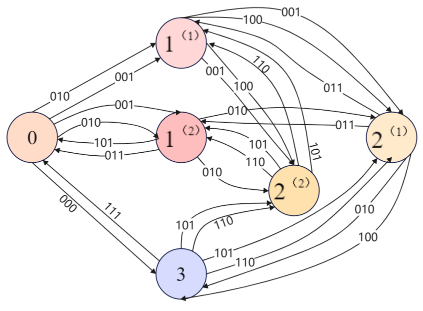

Fig. 3: H', a new labeled state transition graph of G^3 obtained by $\boldsymbol{v}$-consistent partition of the state

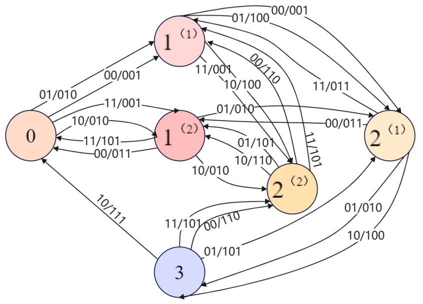

Fig. 4: A finite-state encoder $\boldsymbol{\epsilon}$ with a rate 2:3 for S_0

that the two-bit suffix "11" of $\boldsymbol{x_1}$ is a meaningless block of bits attached in order to decode all source sequences smoothly, while "0111100011" is the actual binary source sequence to be encoded.). Select the initial state $u_0 = 0$ of encoder $\boldsymbol{\epsilon}$. According to Fig. 4, the binary sequences are encoded in 2-bit increments from left to right. From state 0, the edge corresponding to the 2-block input label "01" is transferred to the state $1^{(1)}$, and the output 3-block is labeled "010". Next, starting from the state $1^{(1)}$, the 2-block input label "11" corresponds to the state $2^{(2)}$, and the output 3-block is labeled "001". Repeat this process until the source code is complete. In the end, we get the codeword $\boldsymbol{x_1^1} = 010001110011001101$. It can be verified that this binary codeword satisfies the strong-(4,1)-locally-balanced constraint. Encoder to return to the state sequence is $0, 1^{(1)}, 2^{(2)}, 1^{(2)}, 0, 1^{(2)}, 0$. Then take any length of 18 binary source word $\boldsymbol{y_1} = 010000011010110010$, through Table I, the corresponding quaternary codeword is $\boldsymbol{\sigma} = AGAAACCGTAGCTTCCTC$. It satisfies the strong-(4,1)-locally-GC-balanced constraint.

IV. Decoder

Theorem 1 reveals that the encoder for S, obtained through the state-splitting algorithm discussed above, requires the utilization of a state-dependent decoder. It accepts the q codeword as input and generates a user bit block of length p based on the internal state. This decoder is used to decode sequences that are encoded to satisfy the constraints, and is part of a complete decoding scheme. Here is how it is decoded.

Decoding procedure:

- Initialize the decoder's initial state with the encoder's initial state u_0.

- If the current state is u_i, the current codeword to be decoded and a' future codewords together form a sequence of $a'+1$ blocks (q-block) as the input sequence for the decoder.
- Repeat the second step as long as there are codewords undecoded.

Due to space constraints, see [12] for a detailed explanation. The variable a' in the above decoding step is the FLA of the encoder. Since the graph G is deterministic, its FLA a, is equal to 0. This property is preserved even after the application of powers, meaning the FLA of G^3 is also 0. According to the theory of the state-splitting algorithm, after state-splitting, the FLA increases by 1, resulting in an FLA of a' equal to 1. For any path, the initial state and the first two 3-bit output labels uniquely determine the initial edge, allowing the encoder to be decoded with a delay equal to one block. To overcome this limitation in practice, it is recommended to attach a p-block "11" to the end of the source sequence. This decoder turns the encoder upside down when applied to a valid code sequence, effectively tracing the sequence of states.

Example: Assume that the valid binary sequence in the above example is $\boldsymbol{x_1'} = 0111100011$ of length 10, and "11" is in order to successfully decode the additional meaningless string. We decode the above codeword $\boldsymbol{\sigma}$. The conversion between binary and quaternary results in two binary sequences, $\boldsymbol{x_1^1} = 010001110011001101$ and $\boldsymbol{y_1} = 010000011010110010$, where $\boldsymbol{y_1}$ is source word, but $\boldsymbol{x_1^1}$ also needs to be decoded. The decoder starts in state 0 and the input tag is determined by the combination of the current codeword "010" and a future codeword "001". As a result, the decoder will decode the input tag "01". The decoder then transitions to state $1^{(1)}$. By using "001" and "110", the input tag is confirmed as "00". Keep following this step to decode, and finally, according to the state sequence output by the encoder, the next state is state 0. Then by "001" and the next code word "101" decoded to get a two-bit block "11". The corresponding source sequence is $\boldsymbol{x_1'} = 0111100011$. Comparing $\boldsymbol{x_1'}11 = 011110001111$ to the sequence $\boldsymbol{x_1}$ in the example above shows that the decoding is correct.

The use of the state-dependent algorithm in decoding provides a straightforward and user-friendly approach. However, it is imperative to consider that a state-dependent decoder is susceptible to substantial error propagation when transmitting over noisy channels, as there is no guarantee that the decoder will accurately re-establish the encoder's state. To mitigate this issue, further investigation into the potential of using the slider decoding method as a solution is advisable.

V. Performance Analysis

A. Baselines

To gain insight into the proposed approach, the following encoding schemes are listed below along with ours:

1) **Construction 1** [10]: This encoder employs a one-to-one mapping table of RDS-words and Dyck paths. It encodes each block of size p by analyzing the current RDS word and selecting the corresponding Dyck path from the appropriate table to obtain the codewords.
2) **Construction 2** [10]: The encoder of Construction 2 uses finite state transition diagrams and RDS words.
3) **Our encoder $\boldsymbol{\epsilon}$**: We use the state-splitting algorithm to generate a finite state transition graph, which serves as our encoder $\boldsymbol{\epsilon}$.

B. Comparison of Codebook Sizes

A codebook refers to the set of distinct codewords generated by an encoder starting from its initial state and ending at the same initial state, after reading 3-block output labels across all possible path lengths. For instance, let us consider the encoder $\boldsymbol{\epsilon}$ with an initial state of 0. The codebook of this encoder encompasses all possible paths that initiate and terminate at state 0, while encompassing the number of unique codewords that arise from the reading of 3-block output labels.

We compare the codebook size of our encoder $\boldsymbol{\epsilon}$ with the codebook size of Construction 2. Let $N_1(0,n)$ denote the number of paths in $\boldsymbol{\epsilon}$ that start and end at state 0, with length n. Similarly, $\mathcal{F}_1^n(0)$ represents the number of distinct codewords of length n in $\boldsymbol{\epsilon}$ that start and end at state 0. For Construction 2, we use $N_2(0^+,n)$ to denote the number of distinct paths that start and end at state 0^+, with length n, and $\mathcal{F}_2^n(0^+)$ to represent the number of distinct codewords of length n that start and end at state 0^+. The calculation results are shown in Table II, where n represents the path length, and the corresponding codeword length is $3n$.

TABLE II: Comparison of codebook sizes. The best results are highlighted in bold.

Path length n / **No. paths/codewords**	1	2	3	4	5	6	7	8	...
$N_1(0,n)$	0	1	0	**6**	0	**30**	0	**150**	...
$\mathcal{F}_1^n(0)$	0	4	0	**44**	0	**540**	0	**6114**	...
$N_2(0^+,n)$	0	1	0	2	0	8	0	36	...
$\mathcal{F}_2^n(0^+)$	0	**9**	0	21	0	105	0	963	...

It can be found that the paths obtained by both encoders do not contain paths of odd length. So compare the number of paths with even length. Except for the number of paths of length 2 and the number of codewords of length 6 that are smaller than those of Construction 2, the number of paths and the number of codewords are all larger than those of the encoder of Construction 2. With the increase of n, the number of codewords obtained from $n=4$ keeps increasing, and the growth rate of the number of codewords obtained is 1000%, 1127% and 1032%, respectively. They are larger than the corresponding growth rates of 130%, 400%, and 817% for the number of codewords obtained by the encoder of Construction 2. Moreover, the sum of the number of codewords for $\boldsymbol{\epsilon}$ from 1 to 8 of n gives the number of codewords as 6702, and the number of codewords obtained by the encoder of Construction 2 is 1098. Thus far, the encoder $\boldsymbol{\epsilon}$ gets 6.1 times as many codewords as the encoder of Construction 2. It can be seen from the obtained encoder state transition diagram that the output edge of the state of encoder $\boldsymbol{\epsilon}$ is more dispersed than that of the encoder of Construction 2.

Therefore, it can be inferred that the encoder ϵ can obtain more different codewords of the same length that satisfy the restriction. That is, the codebook obtained by the encoder ϵ is larger than that obtained by the encoder of Construction 2.

C. Comparison of Information Rates

Because the cost of synthesizing DNA is very high, codes should be designed so that nucleotides carry as much information as possible. The information rate is the number of Bits (Bits Per Base, BPB) carried by each base, in bits/nt. The higher the BPB, the better the coding scheme.

In our encoding scheme, the encoder ϵ converts a binary sequence of $2n$ bits (where a valid binary sequence length is only $2n-2$) into a binary sequence of $3n$ bits satisfying the strong-(4,1)-locally-balanced constraint. Then, through the conversion of binary and quaternary in Table I, a DNA sequence consisting of $3n$ bases was obtained by combining with any binary sequence with a length of $3n$. Therefore, it is defined that the BPB of this encoding scheme is $R = \frac{2n-2+3n}{3n} \to 1.667 bits/nt, n \to \infty$. The encoder of Construction 1 encodes any binary information of length sn into a binary sequence of length mn satisfying the strong-(4,1)-locally-balanced constraint ($s : m$ is the encoding rate of the encoder).

Through Table I, combined with any binary sequence with a length of $m*n$, a DNA sequence consisting of mn bases can be obtained. The BPB of this encoding scheme is $R_1 = \frac{s*n+m*n}{mn}$. The calculation results that meet the coding conditions are summarized in the table [10]. It can be found that whether it is $s = 2, m = 4$ or $s = 15, m = 23$, their obtained rates are less than or equal to 0.667. So we know the BPB of the encoder of Construction 1 is $R_1 \leq 1.667 bits/nt$. If the later conditions are ripe, the complexity and efficiency of the coding scheme may be simulated.

D. Comparison of Code Table Sizes

A comparison is made with the encoding table required for the encoder of the latest research scheme [10] Construction 1. In order to make the encoder rate of Construction 1 reach 0.667 and the BPB of the encoding scheme reach 1.667, the input block length should be 12 and the output block length 18. Then the encoding table would need to store $2^{12} = 4096$ items, which is quite impractical. Moreover, the size of the encoding table increases with the length of the input sequence. Our encoder ϵ requires only a table of 24 entries to complete the encoding, which is two orders of magnitude smaller than the encoder required for Construction 1. And the size of the encoding table does not increase with the increase of the input sequence length.

VI. Conclusion

In this paper, we propose encoding and decoding schemes for DNA storage systems that satisfy both 3-RLL and strong-(4,1)-locally-GC-balanced constraints. A finite state encoder ϵ with an encoding rate of 2:3 is designed to satisfy the strong-(4,1)-locally-balanced constrained system by state-splitting algorithm. According to incomplete statistics, the codebook of encoder ϵ is more than 6 times the codebook of existing encoders, which can represent more codewords satisfying strong-(4,1)-locally-balanced constraint. The encoding information rate is greater than that of the existing encoding scheme, and the encoding table used is two orders of magnitude smaller than that of the existing scheme. The state-dependent decoding algorithm is used in the decoding process. The algorithm uses the finite state transition graph obtained by state-splitting to reverse the decoding, which is simple and easy.

Acknowledgment

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. Huaming Wu is the corresponding author.

References

[1] M. G. Ross, C. Russ, M. Costello, A. Hollinger, N. J. Lennon, R. Hegarty, C. Nusbaum, and D. B. Jaffe, “Characterizing and measuring bias in sequence data,” *Genome biology*, vol. 14, no. 5, pp. 1–20, 2013.

[2] P. Yakovchuk, E. Protozanova, and M. D. Frank-Kamenetskii, “Base-stacking and base-pairing contributions into thermal stability of the dna double helix,” *Nucleic acids research*, vol. 34, no. 2, pp. 564–574, 2006.

[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] K. A. S. Immink and K. Cai, “Efficient balanced and maximum homopolymer-run restricted block codes for DNA-based data storage,” *IEEE Communications Letters*, vol. 23, no. 10, pp. 1676–1679, 2019.

[5] J. H. Weber, J. A. M. de Groot, and C. J. van Leeuwen, “On single-error-detecting codes for DNA-based data storage,” *IEEE Communications Letters*, vol. 25, no. 1, pp. 41–44, 2021.

[6] T. T. Nguyen, K. Cai, K. A. S. Immink, and H. M. Kiah, “Capacity-approaching constrained codes with error correction for dna-based data storage,” *IEEE Transactions on Information Theory*, vol. 67, no. 8, pp. 5602–5613, 2021.

[7] K. Cai, Y. M. Chee, R. Gabrys, H. M. Kiah, and T. T. Nguyen, “Correcting a single indel/edit for dna-based data storage: Linear-time encoders and order-optimality,” *IEEE Transactions on Information Theory*, vol. 67, no. 6, pp. 3438–3451, 2021.

[8] Y. Benita, R. S. Oosting, M. C. Lok, M. J. Wise, and I. Humphery-Smith, “Regionalized gc content of template dna as a predictor of pcr success,” *Nucleic acids research*, vol. 31, no. 16, pp. e99–e99, 2003.

[9] R. Gabrys, H. M. Kiah, A. Vardy, E. Yaakobi, and Y. Zhang, “Locally balanced constraints,” in *2020 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2020, pp. 664–669.

[10] C. Wang, Z. Lu, Z. Lan, G. Ge, and Y. Zhang, “Coding schemes for locally balanced constraints,” in *2022 IEEE International Symposium on Information Theory (ISIT)*. IEEE Press, 2022, p. 1342–1347.

[11] K. Fan, H. Wu, and Z. Yan, “Constrained channel capacity for dna-based data storage systems,” *IEEE Communications Letters*, vol. 27, no. 1, pp. 70–74, 2023.

[12] B. H. Marcus, R. M. Roth, and P. H. Siegel, “An introduction to coding for constrained systems,” *Lecture notes*, 2001.

[13] B. H. Marcus, P. H. Siegel, and J. K. Wolf, “Finite-state modulation codes for data storage,” *IEEE Journal on selected areas in communications*, vol. 10, no. 1, pp. 5–37, 1992.

[14] A. D. Weathers and J. K. Wolf, “A new rate 2/3 sliding block code for the,” *IEEE transactions on information theory*, vol. 37, no. 3, pp. 908–913, 1991.

[15] R. Adler, D. Coppersmith, and M. Hassner, “Algorithms for sliding block codes-an application of symbolic dynamics to information theory,” *IEEE Transactions on Information Theory*, vol. 29, no. 1, pp. 5–22, 1983.

[16] P. A. Franaszek, “Construction of bounded delay codes for discrete noiseless channels,” *IBM Journal of Research and Development*, vol. 26, no. 4, pp. 506–514, 1982.

[17] E. Seneta, *Non-negative matrices and Markov chains*. Springer Science & Business Media, 2006.