Design of nonbinary error correction codes with a maximum run-length constraint to correct a single insertion or deletion error for DNA storage

XIAOZHOU LU¹, SUNGHWAN KIM¹,
¹ Department of Electrical, Electronic and Computer Engineering, University of Ulsan, Ulsan 44610, Korea
Corresponding author: Sunghwan Kim (e-mail: sungkim@ulsan.ac.kr).

This work was supported by Samsung Research Funding & Incubation Center for Future Technology (SRFC-IT1802-09).

ABSTRACT Due to the advantages of high information densities and longevity, DNA storage systems have begun to attract a lot of attention. However, common obstacles to DNA storage are caused by insertion, deletion, and substitution errors occurring in DNA synthesis and sequencing. In this paper, we first explain a method to convert binary data into general maximum run-length $r$ sequences with specific length construction, which can be used as the message sequence of our proposed code. Then, we propose a new single insertion/deletion nonbinary systematic error correction code and its corresponding encoding algorithm. For the proposed code, we design the fixed maximum run-length $r$ in the parity sequence of the proposed code to be three. Additionally, the last parity symbol and the first message symbol are always different. Hence, the overall maximum run-length $r$ of the output codeword is guaranteed to be three when the maximum run-length of the message sequence is three. Finally, we determine the feasibility of the proposed encoding algorithm, verify successful decoding when a single insertion/deletion error occurs in the codeword, and present the comparison results with relevant works.

INDEX TERMS DNA storage, maximum run-length, insertion or deletion error, nonbinary systematic error correction code, encoding algorithm.

I. INTRODUCTION

As people gradually rely on more and more data, the hardware of data storage systems has also been gradually upgraded. However, the large space and higher running cost of data centers makes it difficult to meet people’s needs. As a result, researchers have begun to turn their attention to next-generation storage techniques, such as DNA storage. In recent years, DNA storage has gradually been put into practical use due to its longevity, high storage density, and ability to support random access through PCR technology [1]–[3]. For example, Organick et al. stored 200MB of data in 13 million DNA strands [2]. In addition, in [4]–[6], the length of the synthesized DNA strands with four-base nucleotides is 160-200 nt, which helps to meet low-cost requirements.

Although DNA storage has many potential advantages, recent studies [1], [2], [4], [7] have indicated that insertion, deletion, and substitution errors occur in DNA synthesis and sequencing processes. To improve the robustness of the DNA synthesis and sequencing processes, error correction codes such as fountain codes [5], Reed-Solomon codes [7], Bose-Chaudhuri-Hocquenghem codes [8], and low-density parity-check codes [8]–[10] have been applied to DNA storage systems. Since a DNA strand is subject to two constraints, i.e., balanced GC-content (45%–55%) and short homopolymer length (≤3), which enable low error rates during DNA synthesis and sequencing processes [5], the constrained codes [4], [11] without the capability of error correction have been considered to reduce the occurrence of insertion, deletion, and substitution errors in DNA storage. Furthermore, the constrained code combined with the error correction code has been considered for DNA storage [12].

A binary single insertion/deletion error correction (SIDE) code, which is called a Varshamov-Tenengolts (VT) code [13], was first proposed in 1965. Levenshtein modified the binary VT code as an extended VT (EVT) code [14] to correct a single insertion/deletion/substitution error. The nonbinary VT codes [15], which also can correct a single insertion/deletion error, were proposed almost
20 years later. The authors of [16] proposed an efficient systematic encoding algorithm for nonbinary SIDEC codes whose codewords consist of parity symbols and message symbols. A binary EVT code combined with a GC-balanced constrained code [12] and a binary VT code combined with a GC-balanced and \( r \) run-length limited constrained code [17] were applied for DNA storage systems. Additionally, a new binary SIDEC code combined with the maximum run-length \( r \) constrained code and efficient systematic encoding algorithm was proposed in [18].

All of these studies [8], [9], [11], [12], [17] focused on binary coding schemes for DNA storage systems. So far, there have been few studies focusing on \( q \)-ary coding schemes [19] for DNA storage systems. Moreover, insertion and deletion errors are inevitably bound to occur in the process of DNA synthesis and sequencing. For these purposes, we propose a new nonbinary SIDEC code with the maximum run-length \( r \) constraint and systematic encoding algorithm. We can apply the proposed code with \( q = 4 \) and the maximum run-length \( r = 3 \) which are suitable for DNA storage.

Our work is especially inspired by related works [16] and [18]. First, the nonbinary code design in [16] did not consider the maximum run-length constraint and some of the codewords from the design in [16] cannot be successfully encoded. However, these issues are addressed in our code construction. Second, although the binary SIDEC code in [18] involved the maximum run-length \( e \) constraint, the length of output codewords was limited by the maximum run length \( e \). However, our proposed code has no limitations on codewords length. We summary the limitations and weakness of the related works [16], [18] and improvement of our work in Table 1.

We first present a method to convert the binary data into \( q \)-ary symbol sequences with the maximum run-length \( r \) constraint. These sequences can be used as inputs for the proposed nonbinary SIDEC code with the maximum run-length \( r \) constraint. The construction of the proposed \( q \)-ary SIDEC code and its corresponding systematic encoding algorithm are also presented. The output codeword is a sequence of the \( q \)-ary SIDEC code with the maximum run-length \( r \) constraint. Simulation results show that the encoding algorithm is feasible and the \( q \)-ary SIDEC code with the maximum run-length \( r \) constraint can correct a single deletion or insertion error. The main contributions of our work are summarized as follows.

- We propose a new construction of a \( q \)-ary systematic SIDEC code whose codewords meet the maximum run-length \( r \) constraint.
- For the proposed code, we propose systematic encoding algorithm with the parity sequence whose the maximum run-length \( r \) is three.
- The CCF issue is considered in codeword construction and the systematic encoding algorithm.

The paper is organized as follows. In Section II, the notation is given, the proposed code construction and algorithm are introduced, and the method of converting binary data into the maximum run-length \( r \) sequences is presented. In Section III, a nonbinary SIDEC code with the maximum run-length \( r \) constraint and its systematic encoding algorithm are proposed. In Section IV, we briefly explain the decoding method for the proposed nonbinary SIDEC code with the maximum run-length \( r \) constraint. In Section V, we supply the encoding and decoding simulation results and comparison results. Finally, we conclude the paper in Section VI.

### TABLE 1. The limitation and weakness of the related works [16], [18] and improvement of our work

| Limitations and weakness of the related work | Improvement of our work |
|---------------------------------------------|-------------------------|
| - Code construct failure (CCF) issue was not considered. | - The CCF issue is considered in codeword construction and the solution in encoding steps is presented. |
| - Codewords to satisfy the maximum run-length constraint are not guaranteed. | - All output codewords satisfy the maximum run-length constraint. |
| - A single insertion/deletion bit error can be corrected. | - A single nonbinary insertion/deletion symbol can be corrected. |
| From the maximum run length of the binary SIDEC code \( e \leq 7 \), the length of the binary codewords is less than 136. | In our code construction, there is no limitation of codeword length. |

A. NOTATIONS

For any nonnegative integer \( a \) and \( b \), let \([a, b] = \{a + 1, \ldots, b\}\) denote the set \( \{a, a+1, \ldots, b\}\) and \( \{a\} \) for \( a < b \) and \( a = b \), respectively. Moreover, \([a] = [0, a - 1] \). Additionally, \([b]\) denotes the set \( \{0, 1, \ldots, b - 1, b\} \) for \( b \in [a] \). For an integer \( c \), \((a, c, b)\) denotes the vector \( (a, a + c, a + 2c, \ldots, b) \). We only consider \( q \) as \( q = 2^\omega \) for an integer \( \omega > 1 \). Let the four DNA bases, \( A, T, G, \) and \( C \), denote nonbinary symbols 0, 1, 2, and 3 for \( q = 4 \), respectively.

Given a sequence \( v = (v_1, v_2, \ldots, v_n) \in [q]^n \), \( \min(v) \), \( \max(v) \), and \( \text{sum}(v) \) are defined as functions that return the minimum element, maximum element, and summation of all elements in vector \( v \), respectively. The consecutive subsequence \((v_i, v_{i+1}, \ldots, v_j)\) is denoted by \( v_i^j \). The run-length of \( v_i^j \) is \( r = j - i + 1 \) if the symbols in \( v_i^j \) are all the same and they are different from \( v_{i-1} \) and \( v_{j+1} \). The maximum run-length is the length of the longest run in the sequence. Let \( G_b^a \) be the set of sequences \( v \in [q]^b \) of length \( b \) with the maximum run-length \( a \) constraint for any positive integer \( a \) and \( b < a < b \). Here, \([\cdot], [\cdot], \text{mod}, \cap, \log(\cdot)\), and \( \text{dec}_a(\cdot) \) denote the ceiling function, floor function, modulo operation, intersection set, logarithm of base 2, and decimal representation of base \( a \) for any positive integer \( a \), respectively. The Hamming weight of a sequence \( v \) is the number
Our method for constructing the maximum run-length for DNA storage. We segment the with the specified length; this is the message sequence of first symbol of the message sequence. The codewords with last symbol of the parity sequence is always different with the r sequences is similar to the method used in [19]. We extend insertion/deletion error correction and the maximum run-length constraint is similar in [19], we briefly explain constraint is similar in [19], we briefly explain overall block diagram of the proposed code.

\[
\begin{align*}
B. \quad \text{PROPOSED CODE CONSTRUCTION AND ALGORITHM} \\
\end{align*}
\]

The overall block diagram of the proposed code is shown in Fig. 1. Since the construction method that converts the binary data B into a q-ary sequence z with the maximum run-length r constraint is similar in [19], we briefly explain the method in Section II.C. We also present an application of a codeword where the maximum run-length r is three for DNA storage. We segment the q-ary sequence with the maximum run-length r constraint into multiple sequences with the specified length; this is the message sequence of the proposed SIDEC code \( C_{a,b}(n+1) \) in Section III.A. The message sequences used as the inputs pass through the algorithm of the systematic proposed SIDEC code in Section III.C, and the output codewords have properties of single insertion/deletion error correction and the maximum run-length r constraint. The maximum run-length of the output codewords is \( r = 3 \), since we design the fixed maximum run-length \( r \) to be three in message the parity sequences and the last symbol of the parity sequence is always different with the first symbol of the message sequence. The codewords with maximum run-length \( r = 3 \) can be used for DNA synthesis due to the constraints for DNA storage systems [5].

\[
\begin{align*}
C. \quad \text{CONSTRUCTION OF SEQUENCES WITH THE MAXIMUM RUN-LENGTH CONSTRAINT} \\
\end{align*}
\]

Our method for constructing the maximum run-length \( r \) sequences is similar to the method used in [19]. We extend the method in [19] to the general case to obtain the q-ary maximum run-length \( r \) constrained sequence. First, we assume that the binary data B can be divided into \( \mu \) blocks of length \( \epsilon \) bits, which are represented as \( \mathcal{B}^i_{\epsilon} = (\mathcal{B}_{i(1)+1}, \cdots, \mathcal{B}_{i(1)+2}, \cdots) \in [2]^\epsilon \) for \( i \in [1, \mu] \). Then, we build one-to-one mapping to convert \( \mathcal{B}^i_{\epsilon} \) into \((q-1)^{\epsilon-1}\)-ary h elements. Since the sequences described in Section III.A, and the output codewords have properties of single insertion/deletion error correction and the maximum run-length \( r \) constraint, the maximum run-length of the output codewords is \( r = 3 \), since we design the fixed maximum run-length \( r \) to be three in message the parity sequences and the last symbol of the parity sequence is always different with the first symbol of the message sequence. The codewords with maximum run-length \( r = 3 \) can be used for DNA synthesis due to the constraints for DNA storage systems [5].
TABLE 2. The example of one-to-one mapping between a 48-ary element and γ-ary subsequence $z_{(i−1)r+1}^{(r−1)r+1}$ with r symbols.

| 48-ary element | Subsequence $z_{(i−1)r+1}^{(r−1)r+1}$ | 48-ary element | Subsequence $z_{(i−1)r+1}^{(r−1)r+1}$ | 48-ary element | Subsequence $z_{(i−1)r+1}^{(r−1)r+1}$ |
|----------------|------------------------------------|----------------|------------------------------------|----------------|------------------------------------|
| 0              | AAG                               | 14             | TGG                               | 32             | GCA                                |
| 1              | AAC                               | 17             | TGC                               | 33             | GCT                                |
| 2              | ATG                               | 19             | TGA                               | 34             | GTC                                |
| 3              | ATG                               | 20             | TGC                               | 35             | GCG                                |
| 4              | ATG                               | 21             | TCA                               | 36             | CAT                                |
| 5              | ATG                               | 22             | TCA                               | 37             | CAC                                |
| 6              | AGA                               | 23             | TCT                               | 38             | CAC                                |
| 7              | AGT                               | 24             | TGC                               | 39             | CTA                                |
| 8              | AGC                               | 25             | GAT                               | 40             | CTC                                |
| 9              | AGT                               | 26             | GAC                               | 41             | CTC                                |
| 10             | AGC                               | 27             | GTA                               | 42             | GCA                                |
| 11             | AGC                               | 28             | GTG                               | 43             | GCG                                |
| 12             | AGA                               | 29             | GTC                               | 44             | CCA                                |
| 13             | AGT                               | 30             | GCA                               | 45             | CCT                                |
| 14             | ATG                               | 31             | GTC                               | 46             | CTC                                |
| 15             | AGA                               | 32             | TCA                               | 47             | CAC                                |

algorithm of the proposed nonbinary SIDEC code. We design parity symbols that satisfy the following constraints: the maximum run-length r is three and the symbols can be obtained by mapping through tables. The outputs of the systematic encoding algorithm are the maximum run-length r = 3 constraint codewords when the maximum run-length r of the message sequence is not larger than three. Hence, the output codewords have two properties: single insertion/deletion error correction and the maximum run-length r constraint.

A. CODE CONSTRUCTION

The form of the proposed nonbinary systematic code $C_{a,b}(n+1)$ is shown in Table 3. The proposed code is constructed by the sequence $I = (I_1, I_2, \cdots, I_{n+1})$ of length $n+1$, the auxiliary sequence $\alpha = (a_1, a_2, \cdots, a_n)$ of length $n$, and the codeword $c = (c_0, c_1, c_2, \cdots, c_n)$ of length $n+1$. The codeword $c$ is $(p_0, y_1^k)$, which is the concatenation of the parity sequence $p_0^m = (p_0, p_1, \cdots, p_m)$ of length $m + 1$ and the message sequence $y_1^k = (y_1, y_2, \cdots, y_k)$ of length $k$. To determine codeword $c$, monotonically increasing integer sequence $I$ is explained in Definition 1. In Table 3, the subsequence $I_i^r$ of $I$ is used, and $I_{n+1}$ will be used for syndrome calculation later. When the length of message sequence $k$ is given, $I_{n+1}$ should be first determined for the sequence $I$ and the number of the parity symbols can be determined by $I_{n+1}$. However, since $I_{n+1}$ is also recursively calculated after determining $I_i^r$ in Definition 1, it is not possible to determine $I_{n+1}$ and $B = \log(I_{n+1})$ directly. To solve this problem, we first provide the relationship between the message sequence length $k$ and the value of $B$ in Table 4, which can be understood after the sequence generation has ended. We also present the number of parity symbols by (5) and the length of codeword $c$ by (6), as shown in Table 4.

Moreover, since the number of parity symbols depends on whether $B$ is even or odd, and around half of $B$ can make a special pattern in the sequence, we need to define $\sigma$. From $B$, the value of $\sigma$ is defined as

$$\sigma = \left\lceil \frac{B}{2} \right\rceil.$$  

Conversely, from $\sigma$, the value of $B$ can be defined as

$$B = \begin{cases} 2\sigma - 1 & \text{if } B \text{ is an odd number;} \\ 2\sigma & \text{if } B \text{ is an even number}. \end{cases}$$  

(1)

From the relationship between the message sequence length $k$ and the value of $B$ in Table 4, we conclude that

$$k \in [2^{B/2} - 1, 2^{B/2} - 2].$$  

(2)

Definition 1: We define a positive, monotonically increasing integer sequence $I = (I_1, I_2, \cdots, I_{n+1})$ of length $n + 1$. When the length of message sequence $k$ is predefined, the value of $B$ can be determined according to (2). The sequence $I$ is defined as

$$I = (T_1, T_2, \cdots, T_{\sigma-1}, L, I_{m+1}^n),$$  

(3)

where $T_i$ for $i \in [1, \sigma - 1]$, $L$, $m$, $n + 1$, and $I_{m+1}^n$ are expressed as

$$T_i = (2^{2i-2}, 2^{2i-1}, d_{3i}) \quad \text{for } i \in [1, \sigma - 1],$$  

(4)

$$L = \begin{cases} 2^{2\sigma - 2} & \text{for } B = 2\sigma - 1; \\ 2^{2\sigma - 2}, 2^{2\sigma - 1} & \text{for } B = 2\sigma, \end{cases}$$  

$$m = \begin{cases} 3(\sigma - 1) + 2 & \text{for } B = 2\sigma - 1; \\ 3\sigma & \text{for } B = 2\sigma, \end{cases}$$  

(5)

$$n + 1 = m + k + 1,$$  

(6)

$$I_{m+1}^n = \begin{cases} I_m = (2^{2\sigma - 2} + 1, 2^{2\sigma - 1}) & \text{for } B = 2\sigma - 1; \\ 2^{2\sigma - 1} + 1 & \text{for } B = 2\sigma, \end{cases}$$  

(7)

where $d_{3i} \in [2^{2i-1}+1, 2^{2(i+1)-1}-1]$ in (4) for $i \in [1, \sigma - 1]$.

We present some examples of the construction of the positive, monotonically increasing integer sequence $I$. In Example 1, we can intuitively see that the integer sequence $I$ is a monotonically increasing sequence. The reason why some indices in $T_i$ for $i \in [1, \sigma - 1]$ are a multiple of three in (4) is that the fixed maximum run-length $r$ is designed to be three in $p_0^m$ of the proposed code.

Example 1: It is assumed that the lengths of message sequences are $k = 126$ and 145. Then, according to (2), it can be determined that the values of $B$ are 8 and 9, respectively. The values of $m = 12$ and 14, $d_3 \in [3, 3]$, $d_6 \in [9, 15]$, $d_9 \in [33, 61]$, and $d_{12} \in [129, 255]$. Then, the integer sequences $I$ in Definition 1 are given as

$$I = (1, 2, d_3, 4, 8, d_6, 16, 32, d_9, 64, 128, I_m = 129, 130, \cdots, 255, 256) \quad \text{for } k = 126,$$

$$I = (1, 2, d_3, 4, 8, d_6, 16, 32, d_9, 64, 128, d_{12}, 256, I_m = 257, 258, \cdots, 402, 403) \quad \text{for } k = 145.$$
TABLE 3. Forms of the proposed nonbinary systematic SIDECC code.

| $I$ | $I_1$ | $I_2$ | $I_3$ (d3) | $I_3(\sigma - 1)$ (d3,\(\sigma - 1\)) | $I_m$ | $I_m + 1$ | $I_n$ |
|-----|-------|-------|-----------|---------------------------------|-------|--------|-------|
| $\alpha$ | $\alpha_1$ | $\alpha_2$ | $\alpha_3$ | $\alpha_4$ | $\alpha_m$ | $\alpha_m + 1$ | $\alpha_n$ |
| $c$ | $c_0$ | $c_1$ | $c_2$ | $c_3$ | $c_4$ | $c_m$ | $c_m + 1$ | $c_n$ |

TABLE 4. The relationship between the message sequence length k, the value of B, the number of parity symbols $m + 1$, and the length of codeword $e$ + $n$.

| Message sequence length k | $B = \lceil \log (I_n + 1) \rceil$ | $\sigma = \lceil \log \sigma \rceil$ | Number of parity symbols $m + 1$ | Length of codeword $e$ + $n$ |
|---------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|
| $[2^{\beta - 2} - 1, 2^{\beta - 1} - 2]$ | 3 | 2 | 6 | 17, 31, $k + 3\sigma$ |
| $[2^{\beta - 2} - 1, 2^{\beta - 1} - 2]$ | 4 | 2 | 7 | 10, 13, $k + 3\sigma + 1$ |
| $[2^{\beta - 2} - 1, 2^{\beta - 1} - 2]$ | 5 | 3 | 9 | 16, 23, $k + 3\sigma$ |
| $[2^{\beta - 2} - 1, 2^{\beta - 1} - 2]$ | 6 | 3 | 10 | 25, 40, $k + 3\sigma + 1$ |
| $[2^{\beta - 2} - 1, 2^{\beta - 1} - 2]$ | 7 | 4 | 12 | 43, 74, $k + 3\sigma$ |
| $[2^{\beta - 2} - 1, 2^{\beta - 1} - 2]$ | 8 | 4 | 13 | 76, 139, $k + 3\sigma + 1$ |
| $[2^{\beta - 2} - 1, 2^{\beta - 1} - 2]$ | 9 | 5 | 15 | 142, 289, $k + 3\sigma$ |
| $[2^{\beta - 2} - 1, 2^{\beta - 1} - 2]$ | 10 | 5 | 16 | 271, 526, $k + 3\sigma + 1$ |

Definition 2: Consider a sequence $c = (c_0, c_1, c_2, \cdots, c_n) \in [q]^{n+1}$ and its corresponding auxiliary sequence $\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_\sigma) \in [q]^n$, whose element $\alpha_i$ for $i \in [1, n]$ is given by

$$\alpha_i = \begin{cases} 0 & \text{if } c_i < c_{i-1}; \\ 1 & \text{if } c_i \leq c_{i-1}. \end{cases} \quad (8)$$

A syndrome $\text{Syn}(\alpha)$ of the auxiliary sequence $\alpha$ and the check summation sum($c$) of the codeword $c$ are defined as

$$\text{Syn}(\alpha) = \sum_{i=1}^{n} I_i \alpha_i \pmod{I_n + 1}, \quad (9)$$

$$\text{sum}(c) = \sum_{i=0}^{n} c_i \pmod{q}. \quad (10)$$

For $a \in [q]$ and $b \in [I_n + 1]$, the q-ary SIDECC code with parameters $n + 1, a, b$ is defined as

$$C_{a,b}(n + 1) = \{c \in [q]^{n+1} : \text{Syn}(\alpha) = b \pmod{I_n + 1}, \text{sum}(c) = a \pmod{q}\}. \quad (11)$$

Theorem 1: For any codeword $c \in C_{a,b}(n + 1), a \in [q], b \in [I_n + 1]$, the codeword $c$ has the SIDECC property.

Proof: The sequence $I$ is a positive, monotonically increasing sequence. Hence, the sequence $I$ is a monotonically increasing code [20]. Since monotonically increasing codes are also SIDECC codes [20] and sum($c$) = $a \pmod{q}$ satisfies the property of nonbinary VT codes [15], $c \in C_{a,b}(n + 1)$ is a codeword of a SIDECC code.

B. ENCODING ALGORITHM

In this subsection, we present a systematic encoding algorithm for nonbinary SIDECC codes, where the maximum run-length $r$ in the parity sequence is three. To satisfy the run-length constraint between a sequence of parity symbols and a sequence of message symbols, the last parity symbol is designed to be always different from the first message symbol. The output codewords have the properties of nonbinary SIDECC codes and the maximum run-length $r$ which depends on the maximum run-length of the message sequence.

1) Overview

This systematic encoding algorithm converts a message sequence $y^k$ with the maximum run-length $r$ constraint into a SIDECC codeword with the maximum run-length $r$ constraint. The output codeword $c \in C_{a,b}(n + 1) \cap G_{r+1}$ is defined by $c = (p_0^m, y^k)$. The last symbol $p_m$ of $p_0^m$ is used for avoiding the run-length between $p_0^m$ and $y^k$, and $(\sigma - 1)$ symbols of $p_0^m$ whose indices are a multiple of three are used for avoiding the run-length in the parity sequence. The other symbols of $p_0^m$ are represented by the value of $\text{Syn}(\alpha)$.

We explain the encoding algorithm in six steps according to Table 3. The parity sequence $p_0^m$ and message sequence $y^k$ correspond to the partial codeword $c_{0,m} = (c_0, c_1, \cdots, c_m)$ and $c_{m+1} = (c_{m+1}, c_{m+2}, \cdots, c_n)$, respectively.

2) Encoding algorithm steps

Step 1. The length of the message sequence $y^k$ is predefined as $k$. The subsequence $I_1^n$ of $I$ and the value of $I_n + 1$ can be determined by Definition 1. Then, the length $m + 1$ of the parity sequence $p_0^m$ can be obtained by (5) with $I$.

This encoding algorithm requires $d_3i$ for $i \in [1, \sigma - 1], a, b$, and message sequence $y^k$ is $G_r^k$ as input parameters. The output codeword $c \in C_{a,b}(n + 1) \cap G_{r+1}$ is obtained from the proposed encoding algorithm. The symbols $p_{3i}$ are used for avoiding the run-length in the parity sequence $p_0^m$ for $i \in [1, \sigma - 1]$ and the symbol $p_m$ is used for avoiding the run-length between the parity sequence $p_0^m$ and message sequence $y^k$. When the message sequence $y^k$ embeds into the partial codeword $c_{m+1}$, the corresponding partial auxiliary sequence $\alpha_{m+2} = (\alpha_{m+2}, \alpha_{m+3}, \cdots, \alpha_n)$ with length $k - 1$ can be obtained according to (8).

Step 2. Initialize $\alpha_m = 0$, since the parity symbol $c_{m} < c_{m-1}$ in (8) when $\alpha_m = 0$. The constraint $\alpha_m = 0$ prevents an increase in the maximum run-length $r$ between the parity sequence $p_0^m$ and message sequence $y^k$. After initializing $\alpha_m = 0$, we determine the symbol $c_{m} = \min([q] \backslash y_1)$ according to the desirable value of
$c_m$, as shown in Tables 5 and 6. The symbol $c_m$ can be determined as $c_m = \max\{q|y|\}$ according to the desirable value of $c_m$ in Tables 5 and 6 if $\alpha_m$ is flipped from 0 to 1. Meanwhile, the value of $\alpha_m+1$ can be determined after the symbol $c_m$ is determined according to (8). Then, we set all the values of $\alpha_{3i}$ to 1 for $i \in [1, \sigma - 1]$. Due to the condition $c_j < c_{j-1}$ in (8) when $\alpha_j = 0$, if the partial auxiliary sequence $\alpha_{j-1}^T = (\alpha_{j-1}, \alpha_{j+1}, \ldots, \alpha_{j+q-2})$ is $(0, 0, \ldots, 0)$ of length $q$ for $j \in [q, n]$, then the symbol $c_{j-1}$ should be larger than $q - 1$, even though the symbol $c_j = 0$ and the value of symbol $c_{j-1}$ increases by one for $\tau = (j - 1, -1, j - q + 2)$. When $\alpha_j = 1$ and the sequence $\alpha_j^T = (\alpha_j, \alpha_{j+1}, \ldots, \alpha_{j+q-2})$ of length $(q - 1)r$ is $(1, 1, \ldots, 1)$ for $j \in [(q - 1)r, n]$, the symbol $c_{j-1}$ should be less than 0, even though all symbols in $c_j^{T-1}$ are $(c_j - r + 1, c_j - r + 2, \ldots, c_j - r + r)$ are the same as $q - 1$. Additionally, all symbols in sequence $c_j^{T-1-1} = (c_j - r + 1, c_j - r + 2, \ldots, c_j - r + r)$ are the same and the symbol value of $c_j^{T-1-1}$ decreases by one for $\tau = (1, 1, q - 1)$, since the codeword $e$ should satisfy the run-length constraint and the condition $c_j \leq c_{j-1}$ in (8). We define these two symbols, where a symbol should be larger than $q - 1$ or less than zero in the codeword $e$ due to several ones or zeros in the auxiliary sequence $\alpha$, as codeword construct failure (CCF).

Step 3. The remaining bits $(\alpha_1, \alpha_2, \alpha_4, \alpha_5, \ldots, \alpha_{m-1})$ in auxiliary sequence $\alpha$ are used for checking that $\sum(\alpha) = b \pmod{I_{n+1}}$ and its corresponding $I_j$ has a power of 2. Hence, we define $\beta$ as

$$\beta = \sum_{j=1}^{2} \alpha_j \cdot 2^j - 1 + \sum_{i=2}^{q-1} \sum_{j=3}^{2} \alpha_j \cdot 2^j - i$$

$$+ \sum_{j=3}^{q-2} \alpha_j \cdot 2^j - \sigma \pmod{I_{n+1}}$$

$$= b - \sum_{i=3}^{\sigma - 1} \sum_{j=3}^{\sigma - 1} \alpha_j \cdot d_j - \alpha_m I_m$$

$$- k \sum_{j=1}^{\sigma - 1} \alpha_j + m I_{j+m} \pmod{I_{n+1}}.$$  

From the calculation of (12), we can determine the unique remaining bits $(\alpha_1, \alpha_2, \alpha_4, \alpha_5, \ldots, \alpha_{m-1})$ to represent the value of $\beta$.

Step 4. After obtaining the unique auxiliary sequence $\alpha$, we first determine the symbols of $c_m^{\alpha(\sigma-1)} = (c_3(\sigma-1), c_3(\sigma-1)+1, c_m)$ related to $c_m^{\alpha(\sigma-2)} = (c_3(\sigma-2), c_3(\sigma-2)+1, c_m)$ according to Table 5 if the length of $c_m^{\alpha(\sigma-1)}$ is three. If the length of $c_m^{\alpha(\sigma-1)}$ is four, then the length of $c_m^{\alpha(\sigma-1)} = (c_3(\sigma-1), c_3(\sigma-1)+1, c_3(\sigma-1)+2, c_m)$ related to $c_m^{\alpha(\sigma-1)}$ is determined according to Table 6. After the symbols $c_m^{\alpha(\sigma-1)}$ are determined, if CCF occurs, we flip $\alpha_m$.

Step 5. We determine $c_m^{\alpha(\sigma-1)} = (c_3(\sigma-1), c_3(\sigma-1)+1, c_3(\sigma-1)+2, c_3(\sigma-1)+2)$ related to $c_m^{\alpha(\sigma-1)} = (c_3(\sigma-1), c_3(\sigma-1)+1, c_3(\sigma-1)+2, c_3(\sigma-1)+2)$ according to Table 6, for $t = (\sigma - 1, -1, 2)$. After the symbols $c_m^{\alpha(\sigma-1)}$ are determined, if CCF occurs or if the maximum run-length $r > 3$, we flip the current $c_3$. (13)

According to the value of $\gamma$ and the $\alpha_3^{T} = (\alpha_3, \alpha_2, \alpha_1)$, we can determine the symbols $c_m^{\alpha_3}$. Sometimes there is not only one case of $c_m^{\alpha_3}$ that satisfies the value of $\gamma$ and the $\alpha_3^{T}$, we should define the case to avoid the run-length issue as much as possible. Similarly, after the symbols $c_m^{\alpha_3}$ are determined, if CCF occurs or if the maximum run-length $r > 3$, we flip $\alpha_3$.

From (8), the auxiliary sequence $\alpha$ has a strong relationship with the codeword $e$. Obviously, if there are many consecutive ones in $\alpha$, the codeword construction can fail when the codeword should satisfy the maximum run-length constraint. Hence, we first consider $\alpha_m^{\alpha_3(\sigma-1)} = (1, 1, 1, 1)$ in Table 6 for $l \in [1, \sigma]$ and $m = 3\sigma$. Then, there are a total of 19 cases of $c_m^{\alpha_3(l)}$ that correspond to $\alpha_m^{\alpha_3(l)} = (1, 1, 1, 1)$, e.g., $c_m^{\alpha_3(l)} = (c_3(l) - 1, c_3(l), c_3(l)+1, c_3(l)+1)$, $c_m^{\alpha_3(l)} = (c_3(l) - 2, c_3(l), c_3(l)+1, c_3(l)+1)$, etc. Based on the trials, we finally determine $c_m^{\alpha_3(l)} = (c_3(l) - 2, c_3(l) - 1, c_3(l) - 1, c_3(l))$. Similarly, for the case of $\alpha_m^{\alpha_3(l)} = (1, 1, 1, 1)$ in Table 5, we determine its corresponding $c_m^{\alpha_3(l)} = (c_3(l) - c_3(l) - 1, c_3(l) - 1, c_3(l) - 1)$.

For the concatenation issue, the first symbol $c_m^{\alpha_3(l)}$ in $c_m^{\alpha_3(l)}$ will be the last symbol in the partial sequence $c_3^{\alpha_3(l)}$. Refer to the value of $c_3(l)$ in Table 6, which is the last symbol in $c_m^{\alpha_3(l)}$, we can determine the first symbol $c_3^{\alpha_3(l)}$. 

| Table 5. The relationship between $\alpha_3^{\alpha(\sigma-1)}$ and $c_m^{\alpha(\sigma-1)}$ |
|---------------------------------|---------------------------------|---------------------------------|
| $(\alpha_3^{\alpha(\sigma-1)})$ | $(c_m^{\alpha(\sigma-1)})$ | Desired value of $c_m$ |
| $(0, 0, 0)$ | $c_m + 1$ | $c_m + 1$ |
| $(0, 0, 0)$ | $c_m - 1$ | $c_m - 1$ |
| $(0, 0, 0)$ | $c_m$ | $c_m$ |
| $(0, 0, 1)$ | $c_m + 1$ | $c_m + 1$ |
| $(0, 0, 1)$ | $c_m - 1$ | $c_m - 1$ |
| $(0, 0, 1)$ | $c_m$ | $c_m$ |
| $(0, 0, 1)$ | $c_m + 1$ | $c_m + 1$ |
| $(0, 0, 1)$ | $c_m - 1$ | $c_m - 1$ |
| $(0, 0, 1)$ | $c_m$ | $c_m$ |

| Table 6. The relationship between $\alpha_3^{\alpha_3(l)}$ and $c_3^{\alpha_3(l)}$ for $l \in [1, \sigma]$ and $m = 3\sigma$. |
|---------------------------------|---------------------------------|---------------------------------|
| $(\alpha_3^{\alpha_3(l)})$ | $(c_3^{\alpha_3(l)})$ | Desired value of $c_m$ |
| $(0, 0, 0)$ | $c_3(l) + 1$ | $c_3(l) + 1$ |
| $(0, 0, 0)$ | $c_3(l) - 1$ | $c_3(l) - 1$ |
| $(0, 0, 0)$ | $c_3(l)$ | $c_3(l)$ |
| $(0, 0, 1)$ | $c_3(l) + 1$ | $c_3(l) + 1$ |
| $(0, 0, 1)$ | $c_3(l) - 1$ | $c_3(l) - 1$ |
| $(0, 0, 1)$ | $c_3(l)$ | $c_3(l)$ |
| $(0, 0, 1)$ | $c_3(l) + 1$ | $c_3(l) + 1$ |
| $(0, 0, 1)$ | $c_3(l) - 1$ | $c_3(l) - 1$ |
| $(0, 0, 1)$ | $c_3(l)$ | $c_3(l)$ |
as maximum available symbol (i.e., \( q - 1 \)) and minimum available symbol (i.e., 0) when the first bit \( \alpha_{3(l-1)} \) in \( \alpha_{3(l-1)}^{m} \) is 1 and 0, respectively. According to (8), we know that \( c_{l} < c_{l-1} \) if \( \alpha_{l} = 0 \). Hence, when consecutive zeros occur in the partial auxiliary sequence \( \alpha_{3(l-1)}^{m} \), we determine the symbol corresponding to the last zero position in \( \alpha_{3(l-1)}^{m} \) as the minimum available symbol (i.e., 0) and increase the value of each symbol to the left of this symbol by one step by step. The symbol values of \( C_{3(l-1)+1} \) and \( C_{3(l-1)+2} \) are determined by \( \alpha_{3(l-1)+1} \) and \( \alpha_{3(l-1)+2} \). In most cases, we can determine the \( C_{3(l-1)+1} = C_{3(l-1)+2} = q - 1 \) always satisfy \( \alpha_{3(l-1)+1} = \alpha_{3(l-1)+2} = 1 \) according to (8) except \( \alpha_{3(l-1)} = (1, 1, 1, 1) \) and \((0, 1, 1, 1, 1)\). If \( \alpha_{3(l-1)+2} = (0, 1, 0) \), we can determine the symbol value \( C_{3(l-1)+1} \) as \( q = 1 \) to satisfy \( \alpha_{3(l-1)+1} = 1 \) according to (8). On the contrary, we can determine the symbol value \( C_{3(l-1)+1} = 0 \) if \( \alpha_{3(l-1)+2} = (1, 0, 1) \). Similarity, the symbol value \( C_{3(l-1)+2} \) can be determined as \( q = 0 \) if \( \alpha_{3(l-1)+1} = (0, 1, 0) \) and \((1, 0, 1)\), respectively. The value of \( C_{3l(\sigma-1)}^{m(\sigma-1)} \) in Table 5 is determined similarly to \( C_{3l}^{m} \) in Table 6.

C. SPECIAL CASE WITH \( K = 190 \) AND 191 AND THE OVERALL OF ENCODING ALGORITHM

While simulating the encoding algorithm in Section III.B, we find that in most of cases, codewords can be constructed without flipping \( \alpha_{3i} \) for \( i \in [1, \sigma - 1] \) in Steps 5 and 6. However, flipping \( \alpha_{3} \) in Step 6 occurs at two cases where the message sequence length \( k = 190 \) and 191, corresponding to \( I_{n+1} = 433 \) and 434, respectively. In these cases, the codewords are not constructed successfully. When \( \beta = 431 \) in (12) for \( I_{n+1} = 433 \) and 434, the corresponding partial auxiliary sequence is \( \alpha_{3m} = (\alpha_{0}, \alpha_{2}, \ldots , \alpha_{m}) = (1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0) \). Then, we can acquire the partial codeword \( c_{3}^{m} = (c_{1}, c_{4}, \ldots , c_{m}) = (1, 2, 2, 0, 2, 0, 2, 2, 3, 3, c_{m}) \) for \( q = 4 \) according to Tables 5 and 6. We assume that \( \gamma = 3 \) in (13) and the partial codeword \( c_{3}^{m} = (c_{0}, c_{1}, c_{2}) = (1, 1, 1) \) to satisfy \( \gamma = 3 \) when \( \alpha_{3} = (\alpha_{1}, \alpha_{2}, \alpha_{3}) = (1, 1, 1) \). However, since the run-length of \( c_{3}^{m} \) is four, it violates the maximum run-length \( r = 3 \) in parity sequence. According to the encoding algorithm, \( \alpha_{3} \) needs to be flipped from 1 to 0, and the value of \( \beta \) is again calculated as \( \beta = 1 \) and 0. In this case, the corresponding partial auxiliary sequence \( \alpha_{3m} = (1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0) \) and \((0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0)\), respectively. Since there are more than three consecutive zeros in the corresponding partial auxiliary sequence \( \alpha_{3m} \) and \( q = 4 \), the codeword cannot be constructed successfully due to CCF occurs in Step 6. We conclude that these cases are special cases that cannot be constructed successfully, even if the corresponding \( \alpha_{3i} \) is flipped for \( i \in [1, \sigma - 1] \) in Steps 5 and 6.

To address special cases where the codeword cannot be constructed successfully, even if the corresponding \( \alpha_{3i} \) is flipped for \( i \in [1, \sigma - 1] \) in Steps 5 and 6, we make some modifications in Steps 5 and 6. Let \( \delta = (\alpha_{3}, \alpha_{6}, \ldots \, , \alpha_{3(\sigma-1)}) \) be a sequence that consists of the values \( \alpha_{3i} \) in auxiliary sequence \( \alpha \) for \( i \in [1, \sigma - 1] \) and \( \delta_{i} = (\alpha_{3i}, \alpha_{3(i+1)}, \ldots , \alpha_{3(\sigma-1)}) \) for \( i, j \in [1, \sigma - 1] \) and \( i < j \). Notice that the corresponding \( \alpha_{3i} \) in auxiliary sequence \( \alpha \) should also be changed when the value in sequence \( \delta \) has been changed. The modification is that updating \( \text{temp}_{-} \delta = \delta_{i}^{-1} = (\alpha_{3i}, \alpha_{3(i+1)}, \ldots , \alpha_{3(\sigma-1)}) \) and \( \delta_{i}^{-1} = \text{Le}_{\sigma-\text{t}}(\text{Le}_{\sigma-\text{t}}^{-1}(\text{temp}_{-} \delta) - 1) \) until the partial codeword \( c_{3l(\sigma-1)}^{m(\sigma-1)} \) can be constructed successfully for \( t = (\sigma-1, 2) \). For example, from Step 2, the initial sequence \( \delta \) is set as \( \delta = ((1, 1, \cdot \cdot \cdot , 1)) \). If \( c_{3l(\sigma-1)}^{m(\sigma-1)} \) mapping from Table 6 causes CCF to occur or if run-length \( r > 3 \), we set \( \text{temp}_{-} \delta = (1, 1, \cdot \cdot \cdot , 1) \) and update \( \delta_{i}^{-1} = (0, 1, 1, \cdot \cdot \cdot , 1) \), which represent flipping \( \alpha_{3i} \) from 1 to 0. We then go back to Step 3. If \( c_{3l(\sigma-1)}^{m(\sigma-1)} \) causes CCF to occur again or if the run-length still \( r > 3 \), we update \( \text{temp}_{-} \delta = (0, 1, 1, \cdot \cdot \cdot , 1) \) and \( \delta_{i}^{-1} = (1, 0, 1, 1, \cdot \cdot \cdot , 1) \), which represents flipping \( \alpha_{3(i+1)} \) from 1 to 0, and then initialize \( \alpha_{3i} \) (since the codeword cannot be successfully constructed by only flipping \( \alpha_{3i} \)). The update will stop until \( c_{3l(\sigma-1)}^{m(\sigma-1)} \) be constructed successfully. If \( c_{3l(\sigma-1)}^{m(\sigma-1)} \) always causes CCF to occur or if the run-length \( r > 3 \) and \( \text{Le}_{\sigma-\text{t}}^{-1}(\text{temp}_{-} \delta) = 1 \) is less than zero, the codeword cannot be constructed and we define this situation as no code construction (NCC). The modification in Step 6 is similar to the one in the Step 5. The overall encoding algorithm and the accompanying flow chart are summarized by Algorithm 1 and Fig. 2.

According to the encoding algorithm, the codewords are successfully constructed. The simulation results are shown in Section V. In Fig. 2, Y, and N are represented encoding step number, yes, and no, respectively.
Algorithm 1 Encoding the SIDEC code with the maximum run-length $r$ constraint

Input: $d_{si}$ for $i \in [1, \sigma - 1], \alpha$; $b$, and message sequence $y_{ki} \in G_r$.

Output: codeword $c = (p_{0m}, y_{ki}) \in C_{a,b}(n+1) \cap G_r$.

1: Step 1: Embed message sequence $y_{ki}$ in the partial auxiliary sequence $\alpha_{m+2}^n$.
2: Step 2: Set $\alpha_m = 0 \rightarrow c_m = \min(|q|, y_{j1}) \rightarrow \alpha_{m+1}$ and all $\alpha_{3i} = 1$ for $i \in [1, \sigma - 1]$.
3: Step 3: Calculate $\beta$ in (12) to satisfy $\text{Syn}(\alpha) = b \mod (f_{m+1})$.
4: Step 4: $\alpha_{3m}^{(\sigma - 1)} \rightarrow c_{m}^{(\sigma - 1)}$.
5: if the length of $\alpha_{3m}^{(\sigma - 1)}$ is three then
6: determine $c_{m}^{(\sigma - 1)}$ according to $\alpha_{3m}^{(\sigma - 1)}$ with Table 5.
7: else if the length of $\alpha_{3m}^{(\sigma - 1)}$ is four then
8: determine $c_{m}^{(\sigma - 1)}$ according to $\alpha_{3m}^{(\sigma - 1)}$ with Table 6.
9: if $c_{m}^{(\sigma - 1)}$ causes CCF then
10: $\alpha_m \rightarrow c_m = \max(|q|, y_{j1}) \rightarrow \alpha_{m+1}$.
11: goto Step 3.
12: Step 5: $c_{m}^{(\sigma - 1)} \rightarrow c_{m}^{(\sigma - 1)}$.
13: for $t = (\sigma - 1, 1, 2) \text{ do}$
14: determine $c_{m}^{(\sigma - 1)}$ according to $\alpha_{3m}^{(\sigma - 1)}$ with Table 6.
15: if $c_{m}^{(\sigma - 1)}$ causes CCF or run-length $r > 3$ then
16: temp..$\delta = \delta_{\sigma}^{-1}$.
17: if $\text{Le}_{\sigma-1}(\text{temp..}$ $\delta) > 1$ then
18: print (NCC). exit.
19: $\delta_{\sigma}^{-1} = \text{Le}_{\sigma-1}(\text{temp..}$ $\delta) - 1$.
20: goto Step 3.
21: Step 6: Calculate $\gamma$ in (13) to satisfy $\text{sum}(\epsilon) = a \mod (q)$, and $\alpha = c_{00}^2$ according to $\alpha_1^2$.
22: if $c_0^2$ causes CCF or run-length $r > 3$ then
23: temp..$\delta = \delta$.
24: if $\text{Le}_{\sigma-1}(\text{temp..}$ $\delta) > 1$ then
25: print (NCC), exit.
26: $\delta = \text{Le}_{\sigma-1}(\text{temp..}$ $\delta) - 1$.
27: goto Step 3.

IV. DECODING FOR AN INSERTION OR DELETION ERROR

Due to the strong relationship between the auxiliary sequence $\alpha$ and codeword $c$ of $C_{a,b}(n+1)$ code, when a single insertion/deletion error occurs in the codeword $c$ of $C_{a,b}(n+1)$ code, a single insertion/deletion error occurs in the corresponding auxiliary sequence $\alpha$ at the same position. Recall that the sequence $I$ is a monotonically increasing sequence; hence, we can acquire the decoding algorithm by the method used in [21]. Since the $C_{a,b}(n+1)$ code has similar properties as the nonbinary VT codes [15], we can decode for a single insertion/deletion error. Since decoding for an insertion error is similar, we will briefly introduce the decoding method for the deletion error but omit the explanation of decoding for insertion in this paper.

We assume that a deletion error occurs at the position $f \in [1, n]$. From the received codeword $c' = (c_0', c_1', \ldots, c_{f-1}', c_{f+1}', \ldots, c_{n+1}')$ of length $n$, the auxiliary sequence $\alpha' = (\alpha_1', \alpha_2', \ldots, \alpha_{f-1}', \alpha_{f+1}', \ldots, \alpha_n')$ of length $n - 1$ can be recalculated, where $\alpha_f' \in [2]$ and $c_f' \in [q]$. To determine the syndrome, two mappings are defined as

$$L_I^{(0)}(f, \alpha') = \begin{cases} \sum_{j=1}^{f-1} (1 - \alpha_j') \cdot (I_{j+1} - I_j) & \text{for } i \neq 1; \\ 0 & \text{for } i = 1, \end{cases}$$

$$R_I^{(1)}(f, \alpha') = \begin{cases} \sum_{j=1}^{n-1} \alpha_j' \cdot (I_{j+1} - I_j) & \text{for } i \neq n; \\ 0 & \text{for } i = n. \end{cases}$$

Here, $L_I^{(0)}(f, \alpha')$ and $R_I^{(1)}(f, \alpha')$ represent the operations on all zeros to the left of the $f$-th position and on all of the ones to the right of the $f$-th position in the received auxiliary sequence $\alpha'$, respectively. We denote $w_I(\alpha') = R_I^{(1)}(1, \alpha')$, which coincides with the Hamming weight of the received auxiliary sequence $\alpha'$ if the integer sequence $I = (1, 2, \ldots, n + 1)$. The difference in the syndrome values between $\alpha$ and $\alpha'$ is defined as $\Delta = a - \text{Syn}(\alpha') (\mod I_{n+1})$.

The deleted value $\alpha_f'$ is determined by comparing the $\Delta$ value and $w_I(\alpha')$ as

$$\alpha_f' = \begin{cases} 0 & \text{for } \Delta \leq w_I(\alpha'); \\ 1 & \text{for } \Delta > w_I(\alpha'). \end{cases}$$

The sets of possible deletion positions $f$ are $\{ f \in [1, n]; |R_I^{(1)}(f, \alpha') | = \Delta \}$ or $\{ f \in [1, n]; |L_I^{(0)}(f, \alpha') | = \Delta - w_I(\alpha') - I_1 \}$ if the deletion value $\alpha_f' = 0$ or 1, respectively.

After we find the deleted value and possible position of the recalculated auxiliary sequence $\alpha'$, we can infer the deleted value and the possible position in the received codeword $c'$ from the relationship between the deleted symbol $c_f'$ and its left symbol $c_{f-1}'$ according to (8). Similar to the decoding of nonbinary VT codes, we define the deleted symbol $c_f'$ as

$$c_f' = a - \text{sum}(c') \mod (q).$$

With the above method, we can successfully decode a deletion or insertion error, and these results are shown in Section V.

V. SIMULATION AND COMPARISON RESULTS

In this section, we first present the simulation results for our proposed code construction, determine the feasibility of the proposed encoding algorithm, and verify successful decoding when a single insertion/deletion error occurs in the codeword. Then, we present the comparison results between relevant work [16] and our work to present the improvement of our code design.

A. SIMULATION RESULTS OF CODE ENCODING AND DECODING

In this simulation, we determine the values of $q = 4$ and $r = 3$. We segment message sequences $y_{ki}$ into different lengths.
TABLE 7. The encoding results for different codeword lengths with $a = 0$ and $b = 1$.

| Number of sequences | Message length | Codeword length | Successful encoding |
|---------------------|----------------|-----------------|---------------------|
| $1.5 \times 10^8$   | 60             | 72              | ✓                   |
| $1.5 \times 10^8$   | 120            | 133             | ✓                   |
| $1.5 \times 10^8$   | 127            | 138             | ✓                   |
| $1.5 \times 10^8$   | 135            | 150             | ✓                   |
| $1.5 \times 10^8$   | 145            | 160             | ✓                   |
| $1.5 \times 10^8$   | 155            | 170             | ✓                   |
| $1.5 \times 10^8$   | 165            | 180             | ✓                   |
| $1.5 \times 10^8$   | 175            | 190             | ✓                   |
| $1.5 \times 10^8$   | 185            | 200             | ✓                   |
| $1.5 \times 10^8$   | 400            | 415             | ✓                   |

$k = 60, 120, 127, 135, 145, 155, 165, 175, 185,$ and $400$. The reason why the length $k$ is chosen as 127 and 135 is that, even though the two adapters of 24 nt [5] are appended to the head and tail of encoded codeword, respectively, the total length still meets the length of strands in DNA storage, which ranges from 160-200 nt [4]–[6]. Additionally, the length $k$ is chosen from 145 to 185 because the adapters are not considered, and the length of encoded codeword is satisfied the length of strands in DNA storage. Moreover, the reason why we choose $k$ as 60, 120, and 400 is that different values $B$ (i.e., 7, 8, and 10, respectively) need to be considered for general cases. Since the total number of message sequences is very large, we randomly choose $1.5 \times 10^8$ message sequences from the entire message sequences set in order to verify the feasibility of our proposed encoding algorithm. Then, the lengths of the encoded output codewords are 72, 133, 138, 150, 160, 170, 180, 190, 200, and 415, respectively. Since the predefined parameters $a \in [q]$ and $b \in [I_{n+1}]$ have many combinations, we consider two representative scenarios: one where $a = 0$ and $b = 1$ and another where $a$ is a random number in $[q]$ and $b$ is a random number in $[I_{n+1}]$. The results of the successful encoding are listed in Tables 7 and 8.

✓ and X in Tables 7 and 8 denote successful encoding result and encoding failure result, respectively. From these results, the length of 190 codewords corresponds to $I_{n+1} = 433$, and the proposed encoding algorithm can generate the codewords construct successfully.

We assume that a single deletion or insertion error occurs in the received codewords. The position of a single deletion or insertion error is randomly determined and the inserted symbol is randomly chosen from $[q]$ in the received codewords. Then, the decoding algorithm in Section IV is used to correct a single deletion or insertion. The simulation decoding results for single deletion and single insertions are listed in Tables 9 and 10, respectively. ✓ and X in Tables 9 and 10 also denote successful decoding result and decoding failure result, respectively. From Tables 9 and 10, all codewords with a single deletion/insertion are successfully decoded.

**TABLE 8.** The encoding results for different codeword lengths with $a$ is a random number in $[q]$ and $b$ is a random number in $[I_{n+1}]$.

| Number of sequences | Message length | Codeword length | Successful encoding |
|---------------------|----------------|-----------------|---------------------|
| $1.5 \times 10^8$   | 60             | 72              | ✓                   |
| $1.5 \times 10^8$   | 120            | 133             | ✓                   |
| $1.5 \times 10^8$   | 127            | 138             | ✓                   |
| $1.5 \times 10^8$   | 135            | 150             | ✓                   |
| $1.5 \times 10^8$   | 145            | 160             | ✓                   |
| $1.5 \times 10^8$   | 155            | 170             | ✓                   |
| $1.5 \times 10^8$   | 165            | 180             | ✓                   |
| $1.5 \times 10^8$   | 175            | 190             | ✓                   |
| $1.5 \times 10^8$   | 185            | 200             | ✓                   |
| $1.5 \times 10^8$   | 400            | 415             | ✓                   |

**TABLE 9.** The decoding results for different codeword lengths with a single deletion error.

| Number of received sequences | Received codeword length | Successful decoding |
|-----------------------------|--------------------------|---------------------|
| $1.5 \times 10^8$           | 71                       | ✓                   |
| $1.5 \times 10^8$           | 132                      | ✓                   |
| $1.5 \times 10^8$           | 137                      | ✓                   |
| $1.5 \times 10^8$           | 149                      | ✓                   |
| $1.5 \times 10^8$           | 159                      | ✓                   |
| $1.5 \times 10^8$           | 169                      | ✓                   |
| $1.5 \times 10^8$           | 179                      | ✓                   |
| $1.5 \times 10^8$           | 189                      | ✓                   |
| $1.5 \times 10^8$           | 199                      | ✓                   |
| $1.5 \times 10^8$           | 414                      | ✓                   |

**TABLE 10.** The decoding results for different codeword lengths with a single insertion error.

| Number of received sequences | Received codeword length | Successful decoding |
|-----------------------------|--------------------------|---------------------|
| $1.5 \times 10^8$           | 73                       | ✓                   |
| $1.5 \times 10^8$           | 134                      | ✓                   |
| $1.5 \times 10^8$           | 139                      | ✓                   |
| $1.5 \times 10^8$           | 151                      | ✓                   |
| $1.5 \times 10^8$           | 161                      | ✓                   |
| $1.5 \times 10^8$           | 171                      | ✓                   |
| $1.5 \times 10^8$           | 181                      | ✓                   |
| $1.5 \times 10^8$           | 191                      | ✓                   |
| $1.5 \times 10^8$           | 201                      | ✓                   |
| $1.5 \times 10^8$           | 416                      | ✓                   |

**B. COMPARISON RESULTS OF RELATED WORK [16] AND OUR WORK**

Among the related works, the systematic encoding method for nonbinary SIDEC code [16] is most relevant and we provide comparison results with our work. For fair comparison with the related work [16], we still consider the $1.5 \times 10^8$ message sequences with maximum run-length $r = 3$ as inputs, and then make codewords with length $n = 150$ by the systematic encoding method in [16]. The code construction in [16] did not consider the maximum run-length constraint and CCF issues which are caused by some parity symbols $c_{2j}$ located between two message symbols for $j \in [2, \lceil \log_2 n \rceil + 1]$. Then, the parity symbols $c_{2j}$ should be less than zero according to (8) if the message symbol $c_{2j-1} = 0$ and auxiliary bit $c_{2j} = 0$. The ratios of the CCF occurrence and maximum run-length violation in [16] are compared with ones in our work.
TABLE 11. The comparison between [16] and our work for CCF and maximum run-length constraint when the codeword length is $n = 150$.

| CCF only | CCF and maximum run-length constraint violation | Maximum run-length constraint violation | No CCF and no maximum run-length constraint violation |
|----------|---------------------------------------------|-----------------------------|-----------------------------------------------|
| [16]     | 38.66%                                      | 17.33%                      | 21.05%                                        |
| Our work | 0%                                          | 0%                          | 0%                                            |

Among the $1.5 \times 10^8$ outputs codewords from [16], the numbers of codewords which only occur CCF and only violate the maximum run-length constraint are $5,779 \times 10^7$ and $3,158 \times 10^7$, respectively. The number of codewords which occur CCF and violate the run-length constraint simultaneously is $2,599 \times 10^7$, and the number of codewords do not occur CCF and satisfy the maximum run length constraint is $3,442 \times 10^7$. However, the whole codewords of our proposed SIDEc code do not occur CCF and satisfy the maximum run-length constraint. We present the ratios of the CCF occurrence, violation of run-length constraint, and no CCF occurrence and no maximum run-length constraint violation of output codewords with $n = 150$ in Table 11.

From Table 11, about 22.96% codewords of [16] can be used as DNA strands since the CCF do not occur and the maximum run-length constraint is satisfied. Therefore, the outputs codewords in [16] are not suitable for DNA storages. However, the whole codewords from our systematic encoding method for our proposed nonbinary SIDEc code do not occur CCF and satisfy the maximum run-length constraint, which has the potential advantages for DNA storages.

VI. CONCLUSION AND FUTURE WORK

In this work, we propose a nonbinary SIDEc code with the maximum run-length $r$ constraint and its systematic encoding algorithm for the proposed code. Moreover, we verify the feasibility of the encoding and decoding processes of the proposed code. Thanks to the properties of this code design and the error-prone nature of DNA channels, we can apply the proposed code to DNA storage systems. Future work will focus on designing a more efficient encoding process for the nonbinary SIDEc code with the maximum run-length $r$ constraint and constructing codes with the GC-balanced constraint, which is also an important characteristic of DNA storage systems.

REFERENCES

[1] 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, no. 7435, pp. 77–80, Feb. 2013.

[2] L. Organic, S. D Ang, Y. J. Chen, R. Lopez, S. Yekhanin, K. Makarychev, M. Z. Racz, G. Kamath, P. Gopalan, B. Nguyen, C. N. Takahashi, S. Newman, H. Y. Parker, C. Rashchikian, K. Stewart, G. Gupta, R. Carlson, J. Mulligan, D. Carman, G. Seehg, L. Ceze, and K. Strausz, “Random access in large-scale DNA data storage,” Nature Biotechnol., vol. 36, no. 3, pp. 242–248, Mar. 2018.

[3] S. H. T. Yazdi, Y. Yuan, J. Ma, H. Zhao, and O. Milenkovic, “A rewritable, random-access DNA-based storage system,” Sci. Rep., vol. 5, no. 1, p. 14138, Sept. 2015.