Construction and Encoding Algorithm for Maximum Run-Length Limited Single Insertion/Deletion Correcting Code

Reona Takemoto and Takayuki Nozaki
Dept. of Informatics, Yamaguchi University, JAPAN
Email: {a029vbu,tnozaki}@yamaguchi-u.ac.jp

Abstract—Maximum run-length limited codes are constraint codes used in communication and data storage systems. Insertion/deletion correcting codes correct insertion or deletion errors caused in transmitted sequences and are used for combating synchronization errors. This paper investigates the maximum run-length limited single insertion/deletion correcting (RLL-SIDC) codes. More precisely, we construct efficiently encodable and decodable RLL-SIDC codes. Moreover, we present its encoding algorithm and show the redundancy of the code.

I. INTRODUCTION

The error control techniques play an important role to realize the reliable communication systems and data storage systems. Many communication systems and data storage systems employ two types error control techniques, called error correcting codes and constraint codes. The error correcting codes recover the errors caused in the transmitted sequences. The constraint codes give the sequence which is suitable for the specific communication/storage requirements [1].

Synchronization errors cause symbol insertions and symbol deletions in the transmitted sequences. To combat such errors, many insertion/deletion correcting codes have been constructed. In the construction of insertion/deletion correcting codes, there are mainly three approaches, namely, number-theoretic approach, probabilistic approach, and combinatorial approach. In the number-theoretic approach, the codes are defined by single or multiple congruences [2]–[7]. In general, the number-theoretic codes are efficiently decodable and correct a fixed number of insertions/deletions. In the probabilistic approach, the codes are decoded by message passing algorithms and can recover the insertions/deletions caused from statistical channel models [8]–[11]. By the combinatorial approach, we can obtain the code with large number of codewords. However, in general, the most of codes constructed by combinatorial approach are not efficiently decodable [12]. In this paper, we focus on the number-theoretic codes.

Run length of a sequence is the number of the repetition of the same symbols. The $r$-maximum run-length limited ($r$-RLL) code is a constraint code satisfying the maximum run-length of a sequence is smaller than or equal to $r$. It is widely used for the communication and data storage systems, especially, DNA storage system [13].

DNA storage system attracts attention as a future storage system, due to the longevity and high information density. It is reported that the DNA storage system should satisfy the maximum run-length limited and GC-balanced constraints [14], [15]. Immink and Cai constructed such constraint code [16]. Chee et al. gave efficiently encodable GC-balanced code correcting single insertion/deletion/substitution (IDS) [17]. Cai et al. presented efficiently encodable $r$-RLL code correcting single insertion/deletion/substitution (IDS) [18]. However, the code given in [18] suffers long maximum run-length $r$.

This paper constructs efficiently encodable/decodable $r$-RLL codes correcting single insertion/deletion with small $r$. We call such code maximum run-length limited single insertion/deletion correcting (RLL-SIDC) code.

RLL-SIDC codes are also used for the construction of burst-insertion/deletion correcting codes. Schoeny et al. [19] constructed binary $b$-burst insertion or deletion correcting codes, which correct any consecutive insertion or deletion error of length exactly $b$. Non-binary $b$ burst insertion or deletion correcting codes are constructed in [20], [21]. Construction of these codes uses interleaving of codewords, i.e., matrix representation of codewords. The first row of the matrix representation employs a RLL-SIDC code, e.g., RLL-VT code [19]. The other rows employ bounded single insertion/deletion correcting (BSIDC) codes, e.g., shifted VT codes [19], odd coefficient codes [6], or exponential coefficient codes [6]. Nowadays, Lenz and Polianskii proposed efficient binary codes that correct a $b$ or less burst insertion or deletion error [22].

Any encoding algorithm has not been proposed to these burst insertion/deletion correcting codes [20]–[22]. To propose an encoding algorithm to these codes, we need to propose encoding algorithms to RLL-SIDC codes and BSIDC codes. Note that Saeki and Nozaki [23] provided an encoding algorithm for the shifted VT codes.

The purpose of this research is to propose an encoding algorithm for an RLL-SIDC code. To propose an efficient encoding algorithm for the RLL-SIDC code, one might think that we should modify the encoding algorithm of the binary VT code. However, in the systematic encoding algorithm for binary VT codes [24], the parity part is at the positions of 2 powers. The parity part must satisfy the run-length and congruence constraints. Since it is scattered at the positions of 2 powers, it is difficult to satisfy these two constraints at the same time. Therefore, we need to consider another type.
of SIDC code.

Firstly, we construct a systematic-like encodable SIDC code, which has a mechanism to limit the maximum run-length of the codeword. The parity part of the proposed code is consecutive on the front part and consists of two type symbols, namely, symbols to satisfy a constraint defined by a linear congruence.

Secondly, we propose an encoding algorithm for the RLL-SIDC code. It is a variation of modified concatenation [25], [26] in constrained coding. It works in the following procedure: (i) The message is converted into a codeword in the $(0, r − 1)$-constraint code by Wijngaarden and Immink’s algorithm [27, Method C] (WI algorithm); (ii) The $(0, r − 1)$-constraint codeword is transformed into an $r$-RLL sequence by the non-return-to-zero inverted (NRZI); (iii) The encoding algorithm attaches the sequence representing the number of replacement.

This section gives notations used throughout the paper. Table I summarizes possible mapping $\omega_r(x)$ for $x \in [0, 7]$. An $(0, r − 1)$-constraint code is a set of sequences that the run-length of zero symbols are less than $r$. The WI algorithm is known as an encoding algorithm for the $(0, r − 1)$-constraint code. In addition, the NRZI converts a $(0, r − 1)$-constraint codeword to an $r$-RLL codeword.

### B. $r$-RLL codes

An $(0, r − 1)$-constraint code is a set of sequences that the run-length of zero symbols are less than $r$. The WI algorithm is known as an encoding algorithm for the $(0, r − 1)$-constraint code. In addition, the NRZI converts a $(0, r − 1)$-constraint codeword to an $r$-RLL codeword.

#### 1) WI algorithm

Let $G_{r-1}^{(k)}$ be the $(0, r − 1)$-constraint code of length $k$. The WI algorithm converts a binary sequence $u$ of length $k − 1$ into a codeword $x$ in $G_{r-1}^{(k)}$. Here, $k$ satisfies $k \leq 2^r - r - 5$. Roughly speaking, the WI algorithm repeats the replacement step, which removes the forbidden word $0^r 1$ and attaches the sequence representing the position of removed forbidden word, while forbidden word exists. After that, the WI algorithm attaches the sequence representing the number of replacement.

To explain the details of the algorithm, we introduce several notations. We denote substitution $i$ for $j$, by $j \leftarrow i$. The mapping $\omega_r(x) : \{0, 1\}^* \to \{0, 1\}^*$ gives the sequence representing the number $s$ of replacement. The output is defined as follows:

$$\omega_{r-1}(s) := 0^{s-r}v(1^{r-1}0)^v,$$

where $v := \lfloor s/(k+1) \rfloor$ and $\lfloor \cdot \rfloor$ stands for the floor function.

For instance, the output of $\omega_4(x)$ for $x \in [0, 7]$ is summarized in Table I.

| $x$ | $\omega_4(x)$ |
|-----|---------------|
| 0   | 0             |
| 1   | 0             |
| 2   | 0 0011110     |
| 3   | 0000 0011110  |

### II. PRELIMINARIES

This section gives notations used throughout the paper. This section also introduces existing algorithms, namely the WI algorithm and the NRZI, for constructing an encoding algorithm of the RLL codes.

#### A. Notation

Let $\mathbb{Z}$, $\mathbb{Z}^+$ be the set of integers and positive integers, respectively. Let $[a, b]$ be the set of integers between $a$ and $b$, i.e., $[a, b] := \{i \in \mathbb{Z} | a \leq i \leq b\}$. For $Z \subseteq \mathbb{Z}$, denote its minimum and maximum, by $\min Z$ and $\max Z$, respectively. For example, if $Z = [a, b]$, $\min Z = a$ and $\max Z = b$ hold. For $a, b \in \mathbb{Z}$ and $n \in \mathbb{Z}^+$, denote $a \equiv b \pmod n$ if $(a - b)$ divides $n$. Denote the exclusive OR (XOR) by $\oplus$.

Every positive integer $x \in [1, 2^k - 1]$ is represented by $\sum_{i=0}^{k-1} g_i \cdot 2^i$, $g_i \in \{0, 1\}$. For a fixed $k \in \mathbb{Z}^+$, we define $L_k : [0, 2^k - 1] \to \{0, 1\}^k$ as $L_k(x) = g_0g_1g_2 \cdots g_{k-1}$. This mapping is called little-endian of integer. For example, $L_3(4) = 001$ and $L_4(11) = 1101$.

Denote concatenation of sequences $x$ and $y$, by $xy$. Let $\lambda$ be the null string. For sequence $x$ and $i \in \mathbb{Z}^+$, recursively define $x^i = x x^{i-1}$, where $x^0 = \lambda$. The consecutive subsequence $(x_s, x_{s+1}, \ldots, x_t)$ for sequence $x = (x_1, x_2, \ldots, x_n)$ $(1 \leq s < t \leq n)$ is denoted by $x_{[s,t]}$. For $1 < i < j < n$, the subsequence $x_{[i,j]}$ is a run of length $r = j - i + 1$ if $x_{i-1} \neq x_i = x_{i+1} = \cdots = x_j \neq x_{j+1}$. As exceptions, $x_1$ is the start of a run and $x_n$ is the end of a run. The length of the longest run in a sequence is called the maximum run-length. Let $S_{n,r}$ be the set of sequences $x \in \{0, 1\}^n$ whose maximum run-length is smaller or equal to $r$. The code $S_{n,r}$ is called $r$-RLL code of length $n$.
TABLE II

| s | \(v_s\) | \(p_s\) (Linearized) |
|---|---|---|
| 0 | 010001001000100010100000101 | 1 00101 |
| 1 | 0100000101000100001010000101 | 5 01000 |
| 2 | 0100010000100100010100001010 | 10 01101 |
| 3 | 0100010001000101000100010100 | - - |

2) Repeat the conversion so that the run of zero symbols is \(r - 1\) or less. Table II displays this operation. As a result, we get

\[ v_31 = (1010001001010100010001) \].

3) Set \(w \leftarrow 000\) and output \(v_31\) \(w\) as the codeword \(x\):

\[ x = (10100010010100011011000) \].

The decoding algorithm is described in [27].

2) **NRZI:** The NRZI converts a \((0, r - 1)\)-constraint word \(x\) of length \(k - 1\) into \(y \in S_{k,r}\). The encoding algorithm sets \(y_1 = x_1\) and \(y_i = y_{i-1} \oplus x_i\) for \(i \geq 2\).

The decoder of the NRZI converts \(y \in S_{k,r}\) into a \((0, r - 1)\)-constraint word \(x\) of length \(k - 1\). The decoding algorithm sets \(x_1 = y_1\) and \(x_i = y_{i-1} \oplus y_i\) for \(i \geq 2\).

**Example 2:** For the following input output of the NRZI is

\[ x = (10100010010100011011000), \]
\[ y = (11000001111100111011111) \].

**III. COMPARISON OF r-RLL SEQUENCE ENCODERS**

In this section, we will show that the \(r\)-RLL sequence encoder by the WI algorithm and the NRZI is better than the one by Schoeny et al. [19, Appendix B] from the relation between the code length \(n\) and the maximum run-length \(r\).

We give some lemmas to show that.

**Lemma 1:** For a fixed maximum run length \(r\), the code length \(n\) of the \(r\)-RLL sequence by [19, Appendix B] satisfies

\[ n \leq 2^{r-3} + 1 =: G_r. \quad (1) \]

**Proof:** In the method by [19, Appendix B], the maximum run-length \(r\) is satisfying as follows:

\[ r = \lceil \log_2(n - 1) \rceil + 3. \]

From this, we get Eq. (1).

**Lemma 2:** For a fixed maximum run length \(r\), the code length \(n\) of the \(r\)-RLL sequence by the WI algorithm and the NRZI satisfies

\[ n \leq 2^r + r - 5 =: H_r. \quad (2) \]

**Proof:** In the method by the WI algorithm and the NRZI, code length \(n\) is satisfying \(n \leq 2^{r+1} + r - 4\). Moreover, from Sect. II-B2, the maximum run-length becomes just 1 longer. Combining these, we obtain Eq. (2).

**Remark 1:** For \(r \geq 3\),

\[ H_r - G_r = 7 \cdot 2^{r-3} + r - 6 > 0, \]

holds. Hence, for a fixed maximum run-length \(r\), when we use the method by the WI algorithm and the NRZI, the code length \(n\) becomes longer. In other words, for a fixed code length \(n\), to use the method by the WI algorithm and the NRZI, we can make the maximum run-length \(r\) smaller.

From this remark, in this paper, we use the method by the WI algorithm and the NRZI as an \(r\)-RLL sequence encoder.

**IV. EFFICIENT ENCODABLE RLL-SIDC CODE**

In this section, we construct an efficient encodable code correcting an insertion or deletion error and propose its encoding algorithm. Moreover, we show that the outputs of this encoding algorithm are in the \(r\)-RLL code. Furthermore, we compare the redundancy of the proposed encoding algorithm and the lower bound of the redundancy of the optimal RLL-SIDC code.

**A. Code construction**

**Definition 1:** Consider a sequence \(z := (z_1, z_2, \ldots, z_n) \in \{0, 1\}^n\). Suppose \(\hat{r} \geq 4\) and \(d_2 \in [2^r - 2 + 1, 2^r - 4]\). For fixed \(n, \hat{r}\), and \(d_2\), we define integer sequence \(a[n, \hat{r}, d_2, i]_{i=1}^{n+1}\) with length \(n + 1\) as follows:

\[ a[n, \hat{r}, d_2, i] := \begin{cases} 2^{i-1} & (1 \leq i < \hat{r}) \\
    2^{i-2} & (i = \hat{r}) \\
    2^{i-2} & (i = \hat{r} + 1, \hat{r} + 2) \\
    2^{i-2} - 2 + i & (i \in [\hat{r} + 3, n + 1]). \end{cases} \quad (3) \]

To simplify the notation, denote \(a[n, \hat{r}, d_2, i]\) by \(a_i\). Mapping \(\mu : \{0, 1\}^n \rightarrow \mathbb{Z}\) is defined as follows:

\[ \mu(z) := \sum_{i=1}^{n} a[n, \hat{r}, d_2, i] z_i. \]

For \(b \in [0, a_n+1 - 1]\), we define a code

\[ C_b(n, \hat{r}, d_2) := \{ z \in \{0, 1\}^n : \mu(z) \equiv b \pmod{a_n+1} \}. \]

Note that the integer sequence \(a[n, \hat{r}, d_2, i]_{i=1}^{n+1}\) is positive monotonically increasing. We give some examples of integer sequence \(a[n, \hat{r}, d_2, i]_{i=1}^{n+1}\).

**Example 3:** Fix \(d_2 \in [5, 7]\) and \(d_2 \in [9, 15]\). The integer sequences \(a[21, 4, d_4]\) and \(a[35, 3, d_5]\) are

\[ \{a[21, 4, d_4, i]_{i=1}^{22}\} = (1, 2, 4, d_4, 8, 16, 17, 18, \ldots, 30, 31, 32), \]
\[ \{a[35, 3, d_5, i]_{i=1}^{39}\} = (1, 2, 4, 8, d_5, 16, 32, 33, 34, \ldots, 62, 63, 64). \]

The following theorem shows error correcting capability of \(C_b(n, \hat{r}, d_2)\).

**Theorem 1:** For any \(n \in \mathbb{Z}^+, \hat{r} \geq 4, d_2 \in [2^r - 2 + 1, 2^r - 1]\), and \(b \in [0, a_n+1 - 1]\), \(C_b(n, \hat{r}, d_2)\) is an SIDC code.

**Proof:** Recall that \(a[n, \hat{r}, d_2, i]_{i=1}^{n+1}\) is a positive monotonically increasing sequence. Hence, \(a[n, \hat{r}, d_2, i]_{i=1}^{n+1}\) is a monotonically increasing code [28]. Since monotonically increasing codes are SIDC codes [28], \(C_b(n, \hat{r}, d_2)\) is an SIDC code.
The code length of the parity part
\[ \lceil \cdot \rceil \]
where the run-length of a codeword. The other type is used into three stages. At the first stage, other symbols are computed as satisfying the message part. At the second stage, two specific symbols \( k \) and \( r \) are computed as satisfying \( m \equiv b \mod a_{n+1} \). Algorithm 1, whose details will be shown in Section IV-B2, converts \( y \in S_{k,r} \) into \( z \in C_b(n, \hat{r}, d_{\hat{r}}) \cap S_{n,r} \).

The output is represented by \( z = py \), where \( p \) and \( y \) stand for the parity and the message parts, respectively. The symbols of the parity part are divided into two types. One of them limits the run-length of a codeword. The other type is used for satisfying \( \mu(z) \equiv b \mod a_{n+1} \). Algorithm 1 is divided into three stages. At the first stage, \( y \in S_{k,r} \) is embedded in the message part. At the second stage, two specific symbols are decided to limit the run-length of a codeword. At the last stage, other symbols are computed as satisfying \( \mu(z) \equiv b \mod a_{n+1} \).

2) Encoding algorithm: Fix the length of message part \( k \geq 7 \). We decide \( \hat{r} \) as follows:
\[ \hat{r} = \lceil \log_2(k + 2) \rceil, \]
where \( \lceil \cdot \rceil \) stands for the ceiling function. Let \( m \) be the length of the parity part \( p := (p_1, p_2, \ldots, p_{\hat{r}}, p_{\hat{r}+1}, p_{\hat{r}+2}, p_m) \in \{0, 1\}^m \) and \( m \) satisfies
\[ m = \hat{r} + 3. \]
The code length \( n \) satisfies
\[ n = m + k. \]
This algorithm requires parameters \( r \geq \hat{r}, d_{\hat{r}} \in [2^{\hat{r}-2} + 1, 2^{\hat{r}-1} - 1], \) and \( b \in [0, a_{n+1} - 1], \) as input, where \( a_{n+1} = 2^{\hat{r}} + k + 2 \) from Eq. (3).

The input of the algorithm is \( y \in S_{k,r} \). The output of the algorithm is \( z \in C_b(n, \hat{r}, d_{\hat{r}}) \cap S_{n,r} \), where \( z = py \). Symbols \( p_{\hat{r}}, p_m \) are used for limiting the run-length of the codeword. More precisely, symbol \( p_m = y_1 \oplus 1 \) separates the run of the parity part and the message part. Symbol \( p_{\hat{r}} \) limits the run-length of the parity part. Provisionally, set \( p_{\hat{r}} = 0 \). If the run-length of the parity part exceeds the limit, the symbol is changed \( p_{\hat{r}} = 1 \). We will show this ad hoc method always limits the run length of parity part in Sect. IV-D. The other symbols of the parity part are computed for satisfying \( \mu(z) \equiv b \mod a_{n+1} \). Define
\[ q := (p_1, p_2, \ldots, p_{\hat{r}+1}, p_{\hat{r}+2}) \in \{0, 1\}^{\hat{r}+1}. \]

Table III shows \( a_i \) and \( p_i \) for \( i \in [1, m] \). From this table, we see that \( a_i \) is a power of 2 for \( i \in [1, m - 1] \setminus \{\hat{r}\} \). Hence, \( \mu(py) \equiv b \mod a_{n+1} \) is equivalent to
\[ \sum_{i=0}^{\hat{r}} 2^iq_i \equiv b - d_{\hat{r}}p_{\hat{r}} - a_mp_m - \sum_{j=1}^{k} a_{j+m}y_j \mod a_{n+1}. \]

From this, we determine \( q \) uniquely once \( p_{\hat{r}}, p_m \) and \( y \) are decided. Algorithm 1 summarizes the procedure above.

Example 4: For \( y = (10100001000010) \), \( \hat{r} = 4, r = 4, d_4 = 6, b = 31, \) the process of the encoding algorithm is as follows:

1) Embed \( y \) into the message part,
\[ py = (p_1p_2p_3p_{\hat{r}}p_{\hat{r}+1}p_{\hat{r}+2}p_m)_{10100001000010}. \]
2) Set \( p_m = 0 \) and \( p_{\hat{r}} = 0, \)
\[ py = (p_1p_2p_3p_{\hat{r}})_{0101000010000010}. \]
3) Compute \( q \) as satisfy \( \mu(z) \equiv 31 \mod 32, \)
\[ py = (0100000101000001000010). \]
4) Since the maximum run-length of the parity part \( p = (0100000) \) exceed \( r, \) the encoding algorithm resets \( p_{\hat{r}} = 1, \)
\[ py = (p_1p_2p_3p_{\hat{r}})_{0101000010000010}. \]
5) Determine $q$ to satisfy $\mu(py) \equiv b \pmod{a_{n+1}}$ again. Thereafter, output the sequence $py$ as codeword $z$.

$z = (0011110101000010000010)$

Table IV summarizes the change of the parity part.

C. Decoding algorithm

Recall that $C_b(n, \hat{r}, d_r)$ is a monotonically increasing code. Hence, we get decoding algorithm $C_b(n, \hat{r}, d_r)$ by applying [29].

D. Run-length limited property and proof

Theorem 2 shows that Algorithm 1 limits the maximum run-length when $\hat{r} = r$.

**Theorem 2:** Suppose input $y$ is in $S_{k,r}$. If $r = \hat{r} \geq 4$ and $(k, r, \hat{d}_r) \neq (14, 4, 5)$, for all $b \in [0, a_{n+1} - 1]$, the output $z$ of Algorithm 1 satisfies $z \in S_{n,r}$.

**Proof:** For a given $y$, let $z^{(0)}$, $z^{(1)}$ be the sequences obtained by Step 2, 5 of Algorithm 1, respectively. Hence, $z^{(0)}$, $z^{(1)} \in C_b(n, \hat{r}, d_r)$ and $z^{(0)}_{[m+1,n]} = z^{(1)}_{[m+1,n]} = y$ hold. In addition, $z^{(0)}_m = z^{(1)}_m$, $z^{(0)}_r = z^{(1)}_r = 0$, and $z^{(1)}_1 = 1$ also hold.

Since $z^{(0)}_m = z^{(0)}_{m+1}$ and $z^{(1)}_m = z^{(1)}_{m+1}$, the $m$-th and $(m+1)$-th symbols are belong to distinct runs. Moreover, since $y \in S_{k,r}$, the maximum run-length of the message part is smaller than or equal to $r$. Hence, if $\hat{r}$ run-length of the parity part is smaller than or equal to $r$, the maximum run-length of the codeword is also smaller than or equal to $r$. Thereby, we prove using contradiction to be the maximum run-length limited either $p^{(0)}_m := z^{(0)}_{[m+1,n]}$ or $p^{(1)}_m := z^{(1)}_{[m+1,n]}$.

Define mappings $\rho : \{0,1\}^m \to \mathbb{Z}$ and $\sigma : \{0,1\}^k \to \mathbb{Z}$ as follows:

$$\rho(p) := \sum_{i=1}^{m-1} a_i p_i, \quad \sigma(y) := \sum_{i=m+1}^n a_i y_{i-m}.$$ 

Then, the mapping $\mu$ is rewritten by

$$\mu(z) = \rho(p) + dz_z + a_m z_m + \sigma(y). \quad (7)$$

Since $z^{(0)}$, $z^{(1)} \in C_b(n, \hat{r}, d_r)$, we get

$$\mu(z^{(0)}) \equiv b \pmod{a_{n+1}}, \quad \mu(z^{(1)}) \equiv b \pmod{a_{n+1}}.$$ 

This yields

$$\mu(z^{(0)}) - \mu(z^{(1)}) \equiv 0 \pmod{a_{n+1}}.$$
Then, from Table VI, \( A(f_{0,i}, f_{1,j}, d_f) \) takes value in \( B_1(d_f) \cup B_2(d_f) \cup B_3(d_f) \). Recall\( d_f \in [2^f - 2 + 1, 2^f - 1] := K \). Define \( C_i := \bigcup_{d_f \in K} B_i(d_f) \) for \( i = 1, 2, 3 \). Then, we get
\[
\begin{align*}
C_1 &= [2^f - 2, 2^f - 1 - 2], \\
C_2 &= [5 \cdot 2^f - 2 - 3, 3 \cdot 2^f - 1 - 2], \\
C_3 &= [9 \cdot 2^f - 2 - 4, 5 \cdot 2^f - 1 - 2].
\end{align*}
\] (9)
(10)
(11)
Recall that \( a_{n+1} = 2^r + k + 2 \). Since \( \hat{r} = \lfloor \log_2 (k+2) \rfloor \) holds, we get \( k \in [2^f - 1 - 1, 2^f - 2] \) for a given \( \hat{r} \). Define
\[
D := [3 \cdot 2^f - 1 + 1, 2^{r+1}].
\] (12)

Then, \( a_{n+1} \in D \).

For \( l \in Z \) and \( E \subseteq Z \), we define \( lE := \{ le : e \in E \} \). Now, we will give a necessary and sufficient condition for holding
\[
\emptyset = (C_1 \cup C_2 \cup C_3) \cap \left( \bigcup_{l \in Z} lD \right),
\] (13)
i.e., contradicting Eq. (8). Figure 2 depicts the intervals of \( C_1, C_2, C_3, D, \) and \( 2D \). From Eqs. (9), (10), (11), and (12), for \( \hat{r} \geq 4 \), we get
\[
0 < C_1 < C_2 < C_3 < D \leq D, \quad \emptyset = (C_1 \cup C_2 \cup C_3) \cap \left( \bigcup_{l \in Z} lD \right),
\] (13)
(14)
By the proof by contradiction, we obtain that the output \( Z \) of Algorithm 1 is in \( S_{n,r} \) if Eq. (14) holds.

Next, we consider the case of \( z_m^{(0)} = z_m^{(f)} = 1 \). Then the forbidden words are as follows:
\[
\begin{align*}
f_{1,4} &= (1, 1, 1^f - 1, 1, 1), \\
f_{1,5} &= (0, 1, 1^f - 1, 1, 1), \\
f_{1,6} &= (1, 0, 1^f - 1, 1, 1), \\
f_{1,7} &= (0, 0, 1^f - 1, 1, 1), \\
f_{1,8} &= (1, 1, 1^f - 1, 0, 1), \\
f_{0,6} &= (0, 0, 0^f - 1, 0, 1), \\
f_{0,7} &= (1, 0, 0^f - 1, 0, 1), \\
f_{0,8} &= (0, 0, 0^f - 1, 1, 1).
\end{align*}
\]

Table VII gives all the forbidden words and their mapping output \( \rho(f_{ij}) \). In a similar way to the case of \( z_m^{(0)} = z_m^{(f)} = 0 \), we can obtain that the output \( Z \) of Algorithm 1 is in \( S_{n,r} \) if Eq. (14) holds.

**Remark 2:** When \( \hat{r} = 4 \), we should set \( d_f = 6, 7 \) from Theorem 2.

The parameter \( \hat{r} \) determines the sequence \( a_i \), which are the coefficients of code constraint. On the other hand, the parameter \( r \) gives the maximum run-length. Ordinary we set \( r = \hat{r} \). However, we can also set different values satisfying \( r \geq \hat{r} \). Theorem 3 shows that Algorithm 1 limits the maximum run-length when \( r \geq \hat{r} \).

**Theorem 3:** Suppose input \( y \) is in \( S_{k,r} \). If \( \hat{r} \geq 4, \quad r \geq \hat{r}, \) and \( (k, r, d_f) \neq (14, 4, 5) \), for all \( b \in [0, a_{n+1} - 1] \), the output \( z \) of Algorithm 1 also satisfies \( z \in S_{n,r} \).

**Proof:** We have proven the statement in the case of \( r = \hat{r} \). Hence, we should prove the statement for \( r > \hat{r} \). The proof for \( r = \hat{r} + 1 \) can be done in a similar way to \( r = \hat{r} \). The proof for \( r = \hat{r} + 2 \) is trivial since there does not exist any zero and one forbidden words. 

**E. Redundancy**

In this section, we compare the redundancy of the proposed code and a lower bound of the redundancy of the optimal RLL-SIDC code. For a code \( C \) of length \( n \), we define the redundancy \( \mathcal{R}(C) \) as follows:
\[
\mathcal{R}(C) := n - \log_2 |C|,
\]
where \( |C| \) represents the cardinality of a code \( C \). Roughly speaking, the redundancy is the number of additional symbols to encode a message.

Kulkarni and Kiyavash [30] presented an upper bound of the cardinality of the optimal SICD code \( C_{\text{opt}}(n) \) as \( |C_{\text{opt}}(n)| \leq \frac{n^2 - 2}{n - 1} \). Here, the optimal code means the code with the largest cardinality. Moreover, the cardinality of the optimal RLL-SIDC code \( C_{\text{opt}}^{\text{opt}}(n) \) is less than or equal to one of the optimal SICD code, i.e., \( |C_{\text{opt}}^{\text{opt}}(n)| \leq |C_{\text{opt}}(n)| \).

Hence, we get
\[
|C_{\text{opt}}^{\text{opt}}(n)| \leq \frac{n^2 - 2}{n - 1}.
\]
This leads a lower bound of the redundancy of the optimal RLL-SIDC code as follows:
\[
\mathcal{R}(C_{\text{opt}}^{\text{opt}}(n)) \geq n - \log_2 (2n^2 - 2) + \log_2 (n - 1) =: \phi(n).
\] (15)

Let us evaluate the redundancy of the code derived from the proposed encoding algorithm in Sect. IV-B. Recall that
once the code length $n$ is fixed, the parameter $\hat{r}$ is decided by Algorithm 1. More precisely, $\hat{r}$ becomes the smallest positive integer satisfying $n \geq 2^{\hat{r}} + \hat{r} + 1$. Hence, hereafter, we denote the proposed code, by $C_b(n)$, to simplify the notation.

Firstly, we will evaluate the redundancy by using parameter $\hat{r}$. As shown in Fig. 1, the length of message (resp. codeword) is $k = 1$ (resp. $n$). Hence, the redundancy is $n - k + 1$. Combining this and Eqs. (5) and (6), we get
\[ \mathcal{R}(C_b(n)) = \hat{r} + 4. \tag{16} \]

Secondly, we will evaluate the code length $n$ by using parameter $\hat{r}$. Equation (4) leads
\[ k \in [2^{\hat{r} - 1} - 1, 2^\hat{r} - 2]. \]
Combining this condition and Eqs. (5) and (6), we get
\[ n \in [2^{\hat{r} - 1} + \hat{r} + 2, 2^\hat{r} + \hat{r} + 1] =: L_{\hat{r}}. \tag{17} \]
From Eqs. (16) and (17), we obtain the relationship between redundancy and code length.

Theorem 4 shows the difference between the redundancy of the proposed code and the lower bound of the redundancy of the optimal RLL-SIDC code.

**Theorem 4:** For $n \geq L_{\hat{r}} = 14$,
\[ \mathcal{R}(C_b(n)) - \phi(n) < 5. \]
In words, the difference between the redundancy of the proposed code and a lower bound of the redundancy of the optimal RLL-SIDC code is less than 5.

We show a lemma required for proving Theorem 4.

**Lemma 3:** Define $\phi(n)$ as in Eq. (15). For $n \geq 14$, $\phi(n)$ is the monotonically increasing.

**Proof of Lemma 3:** To prove Lemma 3, we show that the derived function $\frac{d\phi(n)}{dn}$ is always positive. The derived function $\phi(n)$ is
\[ \frac{d\phi(n)}{dn} = \frac{2^n - 2(n - 1) \log_2 2}{(2^n - 2)(n - 1) \log_2 2}. \tag{18} \]
Note that $\psi(14) > 0$. For $n \geq 14$, $\psi(n + 1) - \psi(n) = 2^n - 2 \log_2 2 > 0$, holds. Hence, function $\psi(n)$ is always positive. Therefore, for $n \geq 14$, Eq. (18) is always positive. ■

Figure 3 depicts the outline of proof of Theorem 4. Firstly, for a fixed $\hat{r}$, we will show that an upper bound of the difference at $n = 2^{\hat{r}} + 1$ is less than 4. Secondly, we will show $\mathcal{R}(C_b) - \phi(n) < 5$.

**Proof of Theorem 4:** Define
\[ \mathcal{D}(n) := \mathcal{R}(C_b(n)) - \phi(n). \tag{19} \]
Denote
\[ \eta_{\hat{r}} = 2^{\hat{r}} + 1, \tag{20} \]
for a positive integer $\hat{r}$. Note that $\eta_{\hat{r}} \in L_{\hat{r}}$. Equation (20) leads $\hat{r} = \log_2 (\eta_{\hat{r}} - 1)$. Firstly, we evaluate $\mathcal{D}(\eta_{\hat{r}})$ for $\hat{r} \geq 4$. From Eq. (16), the redundancy of the proposed encoding algorithm is as follows:
\[ \mathcal{R}(C_b(\eta_{\hat{r}})) = \log_2 (\eta_{\hat{r}} + 1) + 4. \tag{21} \]
From Eqs. (15), (19), and (21), we get
\[ \mathcal{D}(\eta_{\hat{r}}) = -\eta_{\hat{r}} + \log_2 (2^{\eta_{\hat{r}}} - 2) + 4 < 4. \]
Secondly, for a fixed $\hat{r}$, we discuss the maximum value of $\mathcal{D}(n)$. From Lemma 3 and Eq. (16), for $n \in L_{\hat{r}}$, $\mathcal{D}(n)$ attains its maximum value at
\[ n = L_{\hat{r}} =: \eta'_{\hat{r}}. \]
So, we will calculate $\mathcal{D}(\eta'_{\hat{r}})$. From Eq. (16), $\mathcal{R}(C_b(\eta'_{\hat{r} + 1})) = \mathcal{R}(C_b(\eta_{\hat{r}})) + 1$ holds. Moreover, for $\hat{r} \geq 4$, $\eta_{\hat{r}} < \eta'_{\hat{r} + 1}$ holds. Hence, we obtain $\phi(\eta_{\hat{r}}) < \phi(\eta'_{\hat{r} + 1})$. Combining these, for $\hat{r} + 1 \geq 5$, we get
\[ \mathcal{D}(\eta'_{\hat{r} + 1}) = \mathcal{R}(C_b(\eta'_{\hat{r} + 1})) - \phi(\eta'_{\hat{r} + 1}) \]
\[ = \mathcal{R}(C_b(\eta'_{\hat{r}})) - \phi(\eta'_{\hat{r}}) + 1 \]
\[ < \mathcal{R}(C_b(\eta_{\hat{r}})) - \phi(\eta_{\hat{r}}) + 1 \]
\[ = \mathcal{D}(\eta_{\hat{r}}) + 1 \]
\[ < 5. \]
For $\hat{r} = 4$, we get $\mathcal{D}(\eta'_{4}) \approx 4.299 < 5$. Thus, for $n \geq 14$, $\mathcal{R}(C_b(n)) - \phi(n) < 5$ holds. ■

**V. CONCLUSION**

In this paper, we compare the RLL sequence encoder by the WI algorithm and the NRZI with the one by Schoeny et al. [19, Appendix B]. We proposed an SIDC code which is easily limited the maximum run-length and an encoding algorithm for it. Moreover, we proved that the maximum run-length of the output of the algorithm is limited. Furthermore, we compare the redundancy of the proposed encoding algorithm and the lower bound of the redundancy of the optimal RLL-SIDC code.
We would like to express my gratitude to Dr. Hagiwara at Chiba University to introduce monotonically increasing codes. This research was supported by Inamori Research Grants and Yamaguchi University Fund.

ACKNOWLEDGMENT

REFERENCES

[1] K. A. S. Immink, Codes for mass data storage systems. Shannon Foundation Publisher, 2004.
[2] R. Varshamov and G. Tenengolts, “Codes which correct single asymmetric errors,” Automatika i Telemekhanika, vol. 26, pp. 288–292, 1965.
[3] G. Tenengolts, “Nonbinary codes, correcting single deletion or insertion (corresp.),” IEEE Transactions on Information Theory, vol. 30, no. 5, pp. 766–769, 1984.
[4] V. Levenshtein, “Binary codes capable of correcting deletions, insertions, and reversals,” Soviet physics doklady, pp. 707–710, 1966.
[5] K. Bibak and O. Milenkovic, “Weight enumerators of some classes of deletion correcting codes,” in 2018 IEEE International Symposium on Information Theory (ISIT). IEEE, 2018, pp. 431–435.
[6] T. Nozaki, “Bounded single insertion/deletion correcting codes,” in 2019 IEEE International Symposium on Information Theory (ISIT), June 2019, pp. 2379–2383.
[7] ——, “Weight enumerators for number-theoretic codes and cardinalities of Tenengolts’ non-binary codes,” in 2020 IEEE International Symposium on Information Theory (ISIT). IEEE, 2020, pp. 729–733.
[8] M. C. Davey and D. J. MacKay, “Reliable communication over channels with insertions, deletions, and substitutions,” IEEE Transactions on Information Theory, vol. 47, no. 2, pp. 687–698, 2001.
[9] H. Koremura and H. Kaneko, “Insertion/deletion/substitution error correction by a modified successive cancellation decoding of polar code,” IEICE Trans. Fundamentals, vol. 103, no. 4, pp. 695–703, 2020.
[10] R. Shibata, G. Hosoya, and H. Yashima, “Design and construction of irregular LDPC codes for channels with synchronization errors: New aspect of degree profiles,” IEICE Trans. Fundamentals, vol. 103, no. 10, pp. 1237–1247, 2020.
[11] ——, “Concatenated LDPC/trellis codes: Surpassing the symmetric information rate of channels with synchronization errors,” IEICE Trans. Fundamentals, vol. 103, no. 11, pp. 1283–1291, 2020.
[12] H. Mercier, V. K. Bhargava, and V. Tarokh, “A survey of error-correcting codes for channels with symbol synchronization errors,” IEEE Communications Surveys & Tutorials, vol. 12, no. 1, pp. 87–96, 2010.
[13] S. H. T. Yazdi, H. M. Kiah, E. Garcia-Ruiz, J. Ma, H. Zhao, and O. Milenkovic, “DNA-based storage: Trends and methods,” IEEE Transactions on Molecular, Biological and Multi-Scale Communications, vol. 1, no. 3, pp. 230–248, 2015.
[14] 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, p. R51, 2013.
[15] R. Heckel, G. Mikutis, and R. N. Grass, “A characterization of the DNA data storage channel,” Scientific reports, vol. 9, no. 1, pp. 1–12, 2019.
[16] K. A. S. Immink and K. Cai, “Properties and constructions of constrained codes for DNA-based data storage,” IEEE Access, vol. 8, pp. 49523–49531, 2020.
[17] Y. M. Chee, H. M. Kiah, and T. T. Nguyen, “Linear-time encoders for codes correcting a single edit for DNA-based data storage,” in 2019 IEEE International Symposium on Information Theory (ISIT). IEEE, 2019, pp. 772–776.
[18] K. Cai, X. He, H. M. Kiah, and T. T. Nguyen, “Efficient constrained encoders correcting a single nucleotide edit in DNA storage,” in 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2020, pp. 8827–8830.
[19] C. Schoeny, A. Wächter-Zeh, R. Gabrys, and E. Yaakobi, “Codes correcting a burst of deletions or insertions,” IEEE Transactions on Information Theory, vol. 63, no. 4, pp. 1971–1985, 2017.
[20] C. Schoeny, F. Sala, and L. Dolecek, “Novel combinatorial coding results for DNA sequencing and data storage,” in 2017 51st Asilomar Conference on Signals, Systems, and Computers, Oct 2017, pp. 511–515.
[21] T. Saeki and T. Nozaki, “An improvement of non-binary single b-burst of insertion/deletion correcting code,” IEICE Trans. Fundamentals, vol. E102-A, no. 12, pp. 1591–1599, 2019.
[22] L. Andreas and P. Nikita, “Optimal codes correcting a burst of deletions of variable length,” in 2020 IEEE International Symposium on Information Theory (ISIT). IEEE, 2020, pp. 757–762.
[23] T. Saeki and T. Nozaki, “Systematic encoding algorithms for binary and non-binary shifted VT codes (in japanese),” IEICE technical report, vol. 118, no. 478, pp. 307–312, 2019.
[24] K. A. Abdel-Ghaffar and H. C. Ferreira, “Systematic encoding of the Varshamov-Tenengol’ts codes and the constantin-rao codes,” IEEE Transactions on Information Theory, vol. 44, no. 1, pp. 340–345, 1998.
[25] M. Mansuripur, “Enumerative modulation coding with arbitrary constraints and postmodulation error correction coding for data storage systems,” vol. 1499, pp. 72–86, 1991.
[26] K. A. S. Immink, “A practical method for approaching the channel capacity of constrained channels,” IEEE Transactions on Information Theory, vol. 43, no. 5, pp. 1389–1399, 1997.
[27] A. Wijngaarden and K. Immink, “Construction of maximum run-length limited codes using sequence replacement techniques,” IEEE Transactions on Information Theory, vol. 28, no. 2, pp. 200–207, 2010.
[28] M. Hagiwara, “On ordered syndromes for multi insertion/deletion error-correcting codes,” in 2016 IEEE International Symposium on Information Theory (ISIT). IEEE, 2016, pp. 625–629.
[29] H. Takahashi and M. Hagiwara, “Decoding algorithms of monotone codes and azinv codes and their unified view,” in 2020 International Symposium on Information Theory and Its Applications (ISITA), Oct 2020, pp. 284–288.
[30] A. A. Kulkarni and N. Kiyavash, “Nonasymptotic upper bounds for deletion correcting codes,” IEEE Transactions on Information Theory, vol. 59, no. 8, pp. 5115–5130, 2013.