Abstract—Motivated by applications in DNA-based storage and communication systems, we study deletion and insertion errors simultaneously in a burst. In particular, we study a type of error named $t$-deletion-$s$-insertion-burst ((t, s)-burst for short) which is a generalization of the (2, 1)-burst error proposed by Schoeny et al. Such an error deletes $t$ consecutive symbols and inserts an arbitrary sequence of length $s$ at the same coordinate. We provide a sphere-packing upper bound on the size of binary codes that can correct a ($t$, $s$)-burst error, showing that the redundancy of such codes is at least $\log(n + 2t + 2) - t - 1$. For $t \geq 2s$, an explicit construction of binary ($t$, $s$)-burst correcting codes with redundancy $\log n + (t - 1) \log n + O_t(1)$ is given, where $O_t(\cdot)$ suppresses factors that depend only on $t$. Additionally, we construct a binary ($3$, 1)-burst correcting code with redundancy at most $\log n + 9$, which is optimal up to an additive constant.

Index Terms—DNA storage, error-correcting codes, deletions, insertions, burst error.

I. INTRODUCTION

DNA-BASED storage is a promising direction for future data storage due to its advantages such as storage density and durability [7]. Due to the error behavior in DNA sequences, codes correcting deletions and insertions have recently attracted significant attention. Meanwhile, synchronization loss occurs due to timing uncertainty in communication and storage systems [6], [18], which also leads to deletions and insertions.

The study of error-correcting codes against deletions and insertions dates back to 1965, when Levenshtein [15] showed that the Varshamov-Tenengol’ts (VT) code [26] can correct a single deletion. In the same work, Levenshtein proved the equivalence between $t$-deletion-correcting codes and $t$-insertion-correcting codes, and showed that the redundancy of a $t$-deletion-correcting code is asymptotically at least $t \log n$. It was not until recent years that deletion correcting codes were reconsidered due to advances in DNA storage.

The general problem of correcting $t$ arbitrary deletions has been considered in a series of works [2], [9], [11], [12], [20], [21], [24], [25]. The state-of-the-art result of them is the one from [24], which constructed a $t$-deletion-correcting code with redundancy $(4t - 1) \log n + o(\log n)$.

Both in DNA-based storage and communication systems, the deletion and insertion errors tend to occur in bursts (i.e., consecutive errors) [27]. Therefore, it is of great significance to design codes capable of correcting a burst of deletions/insertions. A code that can correct exactly $t$ consecutive deletions is called a $t$-burst-deletion-correcting code, and a code that can correct at most $t$ consecutive deletions is called a $\leq t$-burst-deletion-correcting code. In 1970, Levenshtein [16] constructed a $\leq 2$-burst-deletion-correcting code, and provided asymptotic bounds on $t$-burst-deletion-correcting codes indicating that the least redundancy to correct $t$ consecutive deletions is asymptotically $\log n + t - 1$. Cheng et al. [4] provided three constructions of $t$-burst-deletion-correcting codes, among which the least redundancy is $t \log(\frac{n}{t} + 1)$. Schoeny et al. [19] constructed a $t$-burst-deletion-correcting code by combining the VT code with constrained coding and utilizing the shifted VT (SVT) code, which has redundancy $\log n + (t - 1) \log n + t - \log t$. As for $\leq t$-burst-deletion-correcting codes, the construction of Schoeny et al. [19] has redundancy $(t - 1) \log n + (\log t - 1) \log n + \log t + O_t(1)$ and then Gabrys et al. [10] reduced the redundancy to $\log t \log n + (\log t - 1) \log n + O_t(1)$. The current best result for $\leq t$-burst-deletion-correcting codes is from Lenz and Polanski [14] with redundancy $\log n + \log t \log n + O_t(1)$. In addition, the model of a burst of $t$ non-consecutive deletions (some $s \leq t$ deletions occur within a block of size $t$) is also considered in [1] and [19].

In addition to deletion or insertion error, substitution error is the most widely considered type of error in classic coding theory. In current DNA storage technology, ultimately one needs to design error-correcting codes against a combination of deletion, insertion, and substitution errors, which is by now a difficult problem [3], [8], [22], [23], [24].

In [19] Schoeny et al. proposed a type of error called a (2, 1)-burst, which deletes two consecutive symbols and then inserts one symbol at the same coordinate. Codes against a (2, 1)-burst error were applied in their model of burst non-consecutive deletions [19]. In this paper, we generalize the error type to a $t$-deletion-$s$-insertion-burst ((t, s)-burst for short), which deletes a burst of $t$ consecutive symbols and then inserts an arbitrary sequence of length $s$ at the same coordinate. Despite being naturally a mixture of deletions and insertions, a $(t, s)$-burst can also be seen as a mixture...
of substitutions and deletions/insertions. For example, when 
\( t \geq s \), a \((t, s)\)-burst can be also seen as deleting a burst of 
\( t - s \) consecutive symbols and then substituting a subset of 
the next \( s \) consecutive symbols. A code that can correct a 
\((t, s)\)-burst error will be called a \( t\)-deletion-\( s\)-insertion-burst 
correcting code \((t, s)\)-burst correcting code for short), and is 
the main objective of this paper.

The rest of the paper is organized as follows. In Section II, 
we give the definitions and some notations used throughout the 
paper and review some previous results that will be used in our 
constructions. In Section III, we prove the equivalence between 
\((t, s)\)-burst correcting codes and \((s, t)\)-burst correcting codes. 
A sphere-packing upper bound on the size of \((t, s)\)-burst 
correcting codes is given in Section IV, which lead to a lower 
bound of the code redundancy. In Section V, we present a 
\((t, s)\)-burst correcting code with redundancy \( \log n + (t - s - 1) \log \log n + O(1) \) for \( t \geq 2s \). Additionally, we present 
an almost optimal \((3, 1)\)-burst correcting code in Section VI.
Lastly, Section VII concludes the paper and discusses some 
future directions.

II. Preliminaries and Related Work

A. Notations and Definitions

Let \( \mathbb{F}_2^n \) be the set of binary sequences of length \( n \). In this 
paper, a sequence \( x \in \mathbb{F}_2^n \) is denoted as either 
\( x_1, x_2, x_3, \ldots, x_n \) or \( \langle x_1, x_2, x_3, \ldots, x_n \rangle \). For integers \( i \leq j \), the length of an 
interval \( [i, j] \) \( \equiv \{i, i + 1, \ldots, j\} \) is defined as \( j - i + 1 \).

A run of \( x = (x_1, x_2, \ldots, x_n) \) is a maximal consecutive 
subsequence of the same symbol. For binary sequences, a run 
is a maximal consecutive 0s or 1s. Let \( R(x) \) denote the run 
sequence of \( x \), and the \( i \)th coordinate of \( R(x) \) is denoted as 
\( R(x)_i = t \) indicates that \( x_i \) lies in the \( t \)th run of \( x \), where \( t \) is counted 
starting from zero. Let \( Rsyn(x) = \sum_{1 \leq i \leq n} R(x)_i \) and let \( r(x) \) be the total 
number of runs in \( x \). For example, if \( x = 10111011000 \), then 
\( r(x) = 4 \).

A \( t \)-deletion-\( s\)-insertion-burst \((t, s)\)-burst for short) error 
on \( x \) is a type of error which deletes \( t \) consecutive symbols 
from \( x \) and inserts an arbitrary sequence of length \( s \) at the 
common coordinate. That is, a \((t, s)\)-burst error over \( x = (x_1, x_2, \ldots, x_n) \) 
starting at the \( i \)th coordinate, \( 1 \leq i \leq n - t + 1 \), will result in 
\( (x_1, x_{i-1}, \ldots, y_1, y_2, \ldots, y_s, x_{i+s}, \ldots, x_n) \), where 
\( (x_1, \ldots, x_{i+s-1}) \) is deleted and \( (y_1, \ldots, y_s) \) is the 
inserted sequence. Continuing with the example above for 
\( x = 10111011000 \), if a \((3, 1)\)-burst error starts at the very 
beginning then one gets either \( 01110000 \) or \( 11110000 \), and 
if a \((1, 3)\)-burst error starts at the very beginning then 
one gets \( y_1 y_2 y_3 \) \( 10111000 \) where \( y_1 y_2 y_3 \) is the inserted sequence. 
Furthermore, a \((t, s)\)-burst error over \( x \) deletes at most 
\( t \) consecutive symbols from \( x \) and inserts an 
arbitrary sequence of length at most \( s \) at the same 
coordinate.

The \((t, s)\)-burst ball of \( x \) is the set of all possible sequences 
obtained by a \((t, s)\)-burst error over \( x \), denoted as \( B_{t,s}(x) \). 
Similarly, we can define the \((\leq t, \leq s)\)-burst ball.

Example 1: Suppose \( x = 101000111 \), \( t = 4 \), \( s = 1 \), then 
\[ B_{4,1}(x) = \{001111, 100111, 110111, 110111, 101011, 101001, 101000 \} \]

A code \( C \subseteq \mathbb{F}_2^n \) is called a \((t, s)\)-burst correcting code if it 
can correct a \((t, s)\)-deletion-\( s\)-insertion-burst error. That is, for any 
two distinct codewords \( c_1, c_2 \in C \), \( B_{t,s}(c_1) \cap B_{t,s}(c_2) = \emptyset \).
A \((\leq t, \leq s)\)-burst correcting code can be defined analogously. 
The redundancy of such a code is given by \( n - \log |C| \).

B. Related Work

In this subsection we briefly review some useful results 
about codes correcting deletions or insertions.

Define the VT syndrome as \( VT(x) = \sum_{i=1}^n ix_i \).
For \( a \in \mathbb{Z}_{n+1} \), the Varshamov-Tenengol’ts (VT) code \( [26] \)
\[ VT_a(n) = \{x \in \mathbb{F}_2^n : VT(x) \equiv a \pmod{n + 1}\} \]
can correct a single deletion. Levenshtein [15] proved that the 
optimal redundancy of \( t \)-deletion-correcting codes is between 
\( t \log n \) and \( 2t \log n \), so the VT code is optimal for a single 
deletion.

In 1970, Levenshtein [16] provided a code that can correct 
a burst of at most 2 deletions as follows:

\[ L_a(n) = \{x \in \mathbb{F}_2^n : Rsyn(0x) \equiv a \pmod{2n} \}, \quad (1) \]

where \( a \in \mathbb{Z}_{2n} \) and the function \( Rsyn(\cdot) \) is applied on \( 0x \), 
the concatenation of a single 0 and the sequence \( x \).

For the \( t \)-burst-deletion-correcting codes, Cheng et al. [4] 
proposed a framework which represents \( x \) as a \( t \times \frac{n}{t} \) array 
(throughout the paper, in an array representation we always 
assume that the number of rows is a divisor of \( n \)) and applies 
some constraints for each row:

\[ A_t(x) = \begin{pmatrix}
  x_1 & x_{t+1} & \cdots & x_{n-t+1} \\
  x_2 & x_{t+2} & \cdots & x_{n-t+2} \\
  \vdots & \vdots & \ddots & \vdots \\
  x_t & x_{2t} & \cdots & x_n
\end{pmatrix} \]

In their constructions the least redundancy is \( t(\log(\frac{n}{t}) + 1) \).

Schoeny et al. [19] followed the framework as in [4] to 
represent a sequence as the array above. In their construction 
the first row is encoded by a variation of the VT code with 
additional run-length constraints, and the other rows apply a 
code called the shifted VT code (SVT):

\[ SVT(n; a, b, P) = \{x \in \mathbb{F}_2^n : VT(x) \equiv a \pmod{P}, \quad \sum_{i=1}^n x_i \equiv b \pmod{2} \} \]

which can correct a single deletion given the additional 
knowledge of the erroneous coordinate within an interval of 
length \( P \) (this knowledge is derived from decoding the first 
row). By choosing \( P = \log(n/t) + 2 \), the redundancy of their 
codes is at most \( \log n + (t - 1) \log \log n + t - \log t \).

In the same paper [19], Schoeny et al. constructed 
a \((2, 1)\)-burst correcting code, which was then applied to
correcting a burst of \( t \) non-consecutive deletions:

\[
C_{2,1}(n; a, b) = \left\{ x \in \mathbb{F}_2^n : V(x) = a \mod 2n - 1, \sum_{i=1}^{n} x_i \equiv b \mod 4 \right\}. \tag{2}
\]

C. Summary of Our Contributions

In this paper, we first prove the equivalence of \((t, s)\)-burst correcting codes and \((s, t)\)-burst correcting codes. Next we compute the size of the error ball for the \((t, s)\)-burst error. Unlike most situations in analyzing deletions/insertions, for \((t, s)\)-burst the size of the error ball is a constant independent of the center and thus it leads to a neat sphere-packing type upper bound, implying that the redundancy of \((t, s)\)-burst correcting codes is at least \( \log(n - t + 2) + t - 1 \). As for constructions, we can focus only on the case \( t \geq s \) due to the aforementioned equivalence. For \( t \geq 2s \), we give a general construction with redundancy \( \log n + (t - s - 1) \log \log n + O_t(1) \), where \( O_t(\cdot) \) suppresses factors that depend only on \( t \). Additionally, we construct a \((3, 1)\)-burst correcting code with redundancy at most \( \log n + 9 \), which is optimal up to an additive constant.

III. EQUIVALENCE OF \((t, s)\)-BURST AND \((s, t)\)-BURST

In this section, we prove the equivalence of \((t, s)\)-burst correcting codes and \((s, t)\)-burst correcting codes. Levenshtein [16] proved the equivalence of \( t \)-deletion-correcting codes and \( t \)-insertion-correcting codes. Schoeny et al. [19] proved the equivalence of codes against a burst of \( t \) deletions and a burst of \( t \) insertions. Using a similar idea, we have the following theorem.

**Theorem 1:** Let \( t \geq s \). A code \( C \) is a \((t, s)\)-burst correcting code if and only if it is an \((s, t)\)-burst correcting code.

**Proof:** We only prove the ‘only if’ part. The ‘if’ part can be proved analogously. Suppose \( C \) is a \((t, s)\)-burst correcting code but not an \((s, t)\)-burst correcting code. Then, there exist two distinct codewords \( x, y \in C \), such that \( B_{s,t}(x) \cap B_{t,s}(y) \neq \emptyset \) is nonempty and thus contains some \( z \in \mathbb{F}_2^{n+1} \). Assume that \( z \) is obtained by deleting \( x_i, \ldots, x_{i+s-1} \) and inserting \( a_1, \ldots, a_t \) at the \( i \)th coordinate of \( x \), and is also obtained by deleting \( y_j, \ldots, y_{j+s-1} \) and inserting \( b_1, \ldots, b_t \) at the \( j \)th coordinate of \( y \). Without loss of generality, we assume \( i \leq j \). Then, \( z \) will have the following two representations:

\[
\begin{align*}
z^1 &= (x_1, \ldots, x_{i-1}, a_1, \ldots, a_t, x_{i+s}, \ldots, x_n), \\
z^2 &= (y_1, \ldots, y_{j-1}, b_1, \ldots, b_t, y_{j+s}, \ldots, y_n).
\end{align*}
\]

1) If \( j \geq i + t \), i.e., the coordinates of deletions of \( y \) are disjoint with the coordinates of insertions of \( x \).

By comparing \( z^1 \) and \( z^2 \) we have

\[
\begin{align*}
(x_1, \ldots, x_{i-1}) &= (y_1, \ldots, y_{i-1}), \\
(a_1, \ldots, a_t) &= (y_{i+1}, \ldots, y_{i+t-1}), \\
(x_{i+s}, \ldots, x_{j+s-1}) &= (y_{i+t}, \ldots, y_j), \\
(x_{j+s}, \ldots, x_n) &= (b_1, \ldots, b_t).
\end{align*}
\]

Therefore, if we delete \((x_j+s-1, \ldots, x_{j+s-1})\) from \( x \) and insert \((y_j, \ldots, y_j+s-1)\) at this coordinate, then we get

\[
(x_1, \ldots, x_{j+s-1}, y_j, \ldots, y_{j+s-1}, x_{j+s}, \ldots, x_n).
\]

If we delete \((y_j+s, \ldots, y_{j+t-1})\) from \( y \) and insert \((x_j, \ldots, x_{j+s-1})\) at this coordinate, then we get

\[
(y_1, \ldots, y_{j-1}, x_j, \ldots, x_{j+s-1}, y_{j+t}, \ldots, y_n).
\]

From the equations above, we have

\[
\begin{align*}
(x_1, \ldots, x_{j+s-1}, y_j, \ldots, y_{j+s-1}, x_{j+s}, \ldots, x_n) \\
= (y_1, \ldots, y_{j-1}, x_j, \ldots, x_{j+s-1}, y_{j+t}, \ldots, y_n).
\end{align*}
\]

Thus, \( B_{t,s}(x) \cap B_{s,t}(y) \neq \emptyset \), which is a contradiction to the hypothesis that \( C \) is a \((t, s)\)-burst correcting code.

2) If \( j \leq i+t-1 \), then the coordinates of deletions of \( y \) and the coordinates of insertions of \( x \) have some intersection. We still have

\[
\begin{align*}
(x_1, \ldots, x_{i-1}) &= (y_1, \ldots, y_{i-1}), \\
(x_{j+s}, \ldots, x_n) &= (y_{j+s}, \ldots, y_n).
\end{align*}
\]

In this case, if we delete \((x_{j-t+1}, \ldots, x_j)\) from \( x \) and insert \((y_j-t+1, \ldots, y_{j-t+s})\) at this coordinate, then we get

\[
(x_1, \ldots, x_{j-t}, y_{j-t+1}, \ldots, y_{j-t+s}, x_{j+1}, \ldots, x_n).
\]

For \( y \), if we delete \((y_{j-t+s+1}, \ldots, y_{j+s})\) and insert \((x_{j+1}, \ldots, x_{j+s})\) at this coordinate, then we get

\[
(y_1, \ldots, y_{j-t+s}, x_{j+1}, \ldots, x_{j+s}, y_{j+s+1}, \ldots, y_n).
\]

Since \( j \leq i+t-1 \), we have \((x_1, \ldots, x_{j-t}) = (y_1, \ldots, y_{j-t}) \). Therefore, one can check that

\[
\begin{align*}
(x_1, \ldots, x_{j-t}, y_{j-t+1}, \ldots, y_{j-t+s}, x_{j+1}, \ldots, x_n) \\
= (y_1, \ldots, y_{j-t+s}, x_{j+1}, \ldots, x_{j+s}, y_{j+s+1}, \ldots, y_n).
\end{align*}
\]

Thus, \( B_{t,s}(x) \cap B_{s,t}(y) \neq \emptyset \), which is again a contradiction to the hypothesis that \( C \) is a \((t, s)\)-burst correcting code.

Note that a \((t \leq s, \leq s)\)-burst correcting code can correct any \((t', s')\)-burst error for \( t' \leq t \) and \( s' \leq s \), then it is also able to correct any \((s', t')\)-burst error. Thus we have the following corollary.

**Corollary 1:** Let \( t \geq s \). A code \( C \) is a \((t \leq s, \leq)\)-burst correcting code if and only if it is a \((t \leq s, \leq)\)-burst correcting code.

In [16] Levenshtein proved a stronger result that \( t \)-deletion-correcting codes and \( t \)-insertion-correcting codes are also equivalent to codes correcting arbitrary \( t_1 \) deletions and \( t_2 \) insertions, as long as \( t_1 + t_2 = t \). It is tempting to ask if a similar result holds for the burst error model: Is there a general equivalence between \((t_1, s_1)\)-burst correcting codes and \((t_2, s_2)\)-burst correcting codes, whenever \( t_1 + s_1 = t_2 + s_2 \)? However, this is not true. For example, consider the following three strings \( x = u001010v \), \( y = u11111v \), and \( z = u011010v \), and \( x \) and \( y \) may belong to a \((3, 1)\)-burst correcting code.
code, but they cannot belong to a $(2,2)$-burst correcting code, since $u11100v ∈ B_{2,2}(x) \cap B_{2,2}(y)$, $y$ and $z$ may belong to a $(2,2)$-burst correcting code, but they cannot belong to a $(3,1)$-burst correcting code, since $u1111v ∈ B_{3,1}(y) \cap B_{3,1}(z)$. Thus $(3,1)$-burst correcting codes and $(2,2)$-burst correcting codes are not equivalent. More generally, we have the following theorem, which states the non-equivalence of $(t_1,s_1)$-burst correcting codes and $(t_2,s_2)$-burst correcting codes, for $t_1 + s_1 = t_2 + s_2$ and $t_1 \notin \{t_2, s_2\}$.

**Theorem 2:** For $t_1 + s_1 = t_2 + s_2$ and $t_1 \notin \{t_2, s_2\}$, there exists a $(t_1,s_1)$-burst correcting code which is not a $(t_2,s_2)$-burst correcting code.

**Proof:** Due to the equivalence of $(t,s)$-burst correcting codes and $(s,t)$-burst correcting codes, without loss of generality we can assume $t_1 > t_2 \geq s_2 > s_1$. Let $x_1 = u00\ldots011\ldots100\ldots0$ and $x_2 = u11\ldots111\ldots111\ldots1v$. For any $y ∈ B_{t_1,s_1}(x_1)$, no matter how we pick the first erroneous coordinate of the $(t_1,s_1)$-burst, in $y$ there will be at least $s_1 + 1$ 0’s left in the middle part between $u$ and $v$. However, any $y ∈ B_{t_1,s_1}(x_2)$ has at most $s_1$ 0’s (which are inserted) between $u$ and $v$. Therefore, $B_{t_1,s_1}(x_1) ∩ B_{t_1,s_1}(x_2) = ∅$, and thus $x_1$ and $x_2$ may belong to a $(t_1,s_1)$-burst correcting code. On the other hand, $u1\ldots1\ldots0\ldots0v ∈ B_{t_2,s_2}(x_1) \cap B_{t_2,s_2}(x_2)$, and thus $x_1$ and $x_2$ cannot belong to a $(t_2,s_2)$-burst correcting code.

IV. AN UPPER BOUND ON THE CODE SIZE

There are several upper bounds on the size of burst-deletion-correcting codes. Levenshtein [16] provided an asymptotic upper bound on the size of $t$-burst-deletion-correcting codes: if $C ⊆ F^n_q$ is a $t$-burst-deletion-correcting code, then $|C| ≤ 2^{n−t+1}/n$, which implies that the redundancy is asymptotically at least $\log n + t − 1$. Schoeny et al. [19] constructed a hypergraph whose vertices are all sequences of $F^{2^m}_2$ and the hyperedges are the $t$-burst-deletion balls of all sequences in $F^n_2$. In this way the problem turns into analyzing the matching number of the hypergraph and they provided an explicit upper bound on the size of $t$-burst-deletion-correcting codes as $|C| ≤ (2^{n-t+1} − 2^l)/(n − 2t + 1)$.

A sphere-packing type upper bound is usually not easy to obtain for most models concerning insertions and deletions, since the size of the corresponding error ball usually depends on the choice of the center. However, in this section we derive an unexpected result that the size of a $(t,s)$-burst ball is in fact a constant independent of the center, and thus give a sphere-packing type upper bound on the size of $(t,s)$-burst correcting codes. To analyze the size of the error ball, we first claim that $B_{t,s}(x)$ can be partitioned into several disjoint parts.

For a $(t,s)$-burst of $x$, let the deleted symbols be $x_i, x_{i+1}, \ldots, x_{i+t-1}$ where $1 ≤ i ≤ n − t + 1$, and the inserted symbols be $y_1, y_2, \ldots, y_s$. Observe that, for example, if the deleted symbols are 10101 and the inserted symbols are 111, then essentially this $(5,3)$-burst can be seen as a $(3,1)$-burst deleting 010 and inserting 1. Following this observation, for $k, \ell ≥ 1$, let $B_{k,\ell}^s(x)$ denote the sequences obtained by a $(k,\ell)$-burst over $x$, where the first and last inserted symbols are different from the first and last deleted symbols, respectively.

In other words, for $k, \ell ≥ 1$,

$$B_{k,\ell}^s(x) = \bigcup_{i=1}^{n-k+1} \left\{ x_1 \ldots x_i-1y_1 \ldots y_{\ell}x_{i+k} \ldots x_n, \text{ where } x_i \neq y_1, x_{i+k-1} \neq y_\ell \right\}.$$  

For example, when $k = 1$ and $\ell ≥ 1$, we have $y_1 = y_\ell \neq x_i$, i.e., we delete a symbol $a ∈ \{0,1\}$ and insert a sequence of length $\ell$ starting and ending with $1 − a$. Similarly, when $\ell = 1$ and $k ≥ 1$, we have $x_i = x_{i+k-1} \neq y_1$, i.e., we delete a sequence of length $k$ starting and ending with the symbol $a$ and insert $1 − a$. Moreover, when $k = 0$ or $\ell = 0$, we define $B_{0,\ell}^s(x)$ or $B_{0,0}^s(x)$ simply as the burst insertion ball or the burst deletion ball of $x$, i.e., $B_{0,\ell}^s(x) = B_{0,0}^s(x)$ and $B_{0,0}^s(x) = B_{0,0}^s(x)$. In particular, $B_{0,0}^s(x) = \{x\}$ contains the sequence $x$ itself.

**Lemma 1:** When $t ≥ s$, $B_{t,s}(x)$ is a disjoint union of $B_{t-s-t}^s(x)$, $0 ≤ t ≤ s$. Similarly, when $t ≤ s$, $B_{t,s}(x)$ is a disjoint union of $B_{t-s-t}^s(x)$, $0 ≤ t ≤ s$.

**Proof:** We only prove the first part. The second part can be proved analogously.

First we need to prove that $\bigcup_{t=s}^{n} B_{t-s-t}^s(x) = B_{t,s}(x)$. Obviously, $\bigcup_{t=s}^{n} B_{t-s-t}^s(x) ⊆ B_{t,s}(x)$. For the other direction, consider $w ∈ B_{t,s}(x)$ and suppose $w$ is obtained by deleting $x_i, x_{i+1}, \ldots, x_{i+t-1}$ from $x$ and inserting $y_1, y_2, \ldots, y_s$. Let $p ∈ \{1,2,\ldots,s\}$ be the smallest index such that $y_p \neq x_{i+p-1}$, and let $q ∈ \{1,2,\ldots,s\}$ be the largest index such that $y_q \neq x_{i+q+s-1}$.

- If such $p$ and $q$ exist and $p ≤ q$, then $w$ can be seen as deleting $x_{i+p-1}, \ldots, x_{i+t-1}$ and inserting $y_p, \ldots, y_q$. Thus $w ∈ B_{t-s+p-1,q-p+1}^s(x)$, where $q-p+1 ∈ \{1,2,\ldots,s\}$.
- If $p$ does not exist, it means that $(y_1,\ldots,y_s) = (x_{i+1},\ldots,x_{i+s-1})$, i.e., the inserted sequence is a prefix of the deleted sequence. Then $w$ can be seen as deleting $(x_{i+1},\ldots,x_{i+t-1})$ and thus $w ∈ B_{t-s,0}^s(x)$.
- If $q$ does not exist, it means that $(y_1,\ldots,y_s) = (x_{i+t-1},\ldots,x_{i+s-1})$, i.e., the inserted sequence is a suffix of the deleted sequence. Then $w$ can be seen as deleting $(x_{i+1},\ldots,x_{i+t-1})$ and thus $w ∈ B_{t-s,0}^s(x)$.

Finally, if $p$ and $q$ exist and $p > q$, then it means that $(y_1,\ldots,y_{p-1})$ is a prefix of the deleted sequence and $(y_{p+1},\ldots,y_s)$ is a suffix of the deleted sequence. Then $w$ can be also seen as deleting $(x_{i+p-1},\ldots,x_{i+p+t-s-2})$ and thus $w ∈ B_{t-s,0}^s(x)$.

Therefore, $\bigcup_{t=s}^{n} B_{t-s-t}^s(x) = B_{t,s}(x)$ holds and it only remains to show the disjointness of $B_{t-s-t}^s(x)$ for $0 ≤ t ≤ s$. Suppose there exist a sequence $z ∈ B_{t-s-t}^s(x) \cap B_{t-s-t}^s(x)$, and $z$ is obtained by deleting $t-s+\ell_1$ (resp. $t-s+\ell_2$) symbols and inserting $\ell_1$ (resp. $\ell_2$) symbols at the $i_1$th (resp. $i_2$th) coordinate of $x$.

If $i_1 = i_2$, without loss of generality, we can assume $i_1 < i_2$. Then $z$ has the following two representations:

$$z^1 = (x_1,\ldots,x_{i-1},y_1^1,\ldots,y_{\ell_1}^1,x_{i+t-s+\ell_1+1},\ldots,x_n),$$
$$z^2 = (x_1,\ldots,x_{i-1},y_1^2,\ldots,y_{\ell_2}^2,x_{i+t-s+\ell_2+1},\ldots,x_n).$$
Comparing the \((i + t - s + \ell_2)\)th symbol, we have
\(x_{i+t-s+\ell_2} = y_{\ell_2}^2\). However, since \(z \in B'_{t-s+\ell_2,\ell_2}(x)\), in the representation of \(z^2\) the last deleted symbol \(x_{i+t-s+\ell_2}\) should be different from the last inserted symbol \(y_{\ell_2}^2\), a contradiction.

If \(i_1 \neq i_2\), without loss of generality let \(i_1 < i_2\). Comparing the \(i_1\)th symbol of \(z^1\) and \(z^2\), we have \(y_{i_1} = x_{i_1}\). However, since \(z \in B'_{t-s+\ell_1,\ell_1}(x)\), in the representation of \(z^1\) the first deleted symbol \(x_{i_1}\) should be different from the first inserted symbol \(y_{i_1}\), again a contradiction.

Thus, \(B'_{t-s+\ell_1,\ell_1}(x)\) and \(B'_{t-s+\ell_2,\ell_2}(x)\) are disjoint for any \(\ell \neq \ell_2\), and the lemma is proved.

Next, we need to calculate the size of each \(B'_{k,\ell}(x)\).

When \(\ell = 0\), in [16] Levenshtein calculated the size of a \(k\)-burst-deletion ball as:
\[
|B'_{k,0}(x)| = 1 + \sum_{i=1}^{k} (r(A_k(x)_i) - 1),
\]
where \(k \geq 1\), \(A_k(x)_i\) is the \(i\)th row of the array representation \(A_k(x)\), and \(r(A_k(x)_i)\) is the number of runs in the \(i\)th row. Further note that for the case \(k = \ell = 0\) we have \(B'_{0,0}(x) = \{x\}\) and thus \(|B'_{0,0}(x)| = 1\), which can also be seen as a special case of Equation (3). In the next lemmas, we will compute the size of \(B'_{k,\ell}(x)\) for the remaining cases.

Lemma 2: Let \(x \in \mathbb{F}_2^n\), \(k \geq 1\), \(\ell = 1\), then
\[
|B'_{k,1}(x)| = n - \sum_{i=1}^{k-1} (r(A_{k-1}(x)_i)).
\]

Proof: First consider \(k \geq 2\). Recall that any \(y \in B'_{k,1}(x)\) is obtained from \(x\) by deleting a string of length \(k\) starting and ending with the same symbol \(a\) and inserting \(1 - a\). Let \(y_1, y_2 \in B'_{k,1}(x)\) where the \((k,1)\)-bursts start at the \(i_1\)th and \(i_2\)th coordinate (WLOG, assume \(i_1 < i_2\)). That is, \(y_1 = (x_1, \ldots, x_{i_1-1}, y_{i_1}, x_{i_1+1}, \ldots, x_n)\) and \(y_2 = (x_1, \ldots, x_{i_2-1}, y_{i_2}, x_{i_2+1}, \ldots, x_n)\). Note that since \(x_{i_1} = x_{i_2}+1 \neq y_{i_1}\) and \(y_{i_2}\) and \(y_2\) have distinct symbols on their \(i_1\)th coordinate and thus \(y_{i_1} \neq y_{i_2}\). Therefore, \(|B'_{k,1}(x)|\) is exactly the number of coordinates \(i\) such that \(x_{i} = x_{i+k-1+1}, 1 \leq i \leq n - k + 1\).

Write \(x\) as a \((k-1) \times \frac{n-k}{\ell+1}\) array:
\[
A_{k-1}(x) = \begin{bmatrix}
    x_1 & x_k & \cdots & x_{n-k+2} \\
    x_2 & x_{k+1} & \cdots & x_{n-k+3} \\
    \vdots & \vdots & \ddots & \vdots \\
    x_{k-1} & x_{2k-2} & \cdots & x_n
\end{bmatrix}.
\]

In the representation of \(A_{k-1}(x)\), \(x_{i}\) and \(x_{i+k-1}\) are two consecutive symbols in a row. The \(i\)th row contributes \(\frac{x_{i+k-1} - r(A_{k-1}(x)_i)}{\ell+1}\) to the size of \(|B'_{k,1}(x)|\). Therefore, for \(k \geq 2\) we have
\[
|B'_{k,1}(x)| = \sum_{i=1}^{k-1} \left( \frac{n - k - 1 - r(A_{k-1}(x)_i)}{\ell+1} \right) = n - \sum_{i=1}^{k-1} r(A_{k-1}(x)_i).
\]

Lemma 3: Let \(x \in \mathbb{F}_2^n\), \(k \geq 1\), \(\ell \geq 2\), then
\[
|B'_{k,\ell}(x)| = (n - k + 1) \cdot 2^{\ell-2}.
\]

Proof: All the sequences in \(B'_{k,\ell}(x)\) have the form \(x_1 \ldots x_{i-1}y_1 \ldots y_{t-s}x_{i+k-1} \ldots x_n\), where \(1 \leq i \leq n-k+1\) is the starting coordinate of the deletion. We claim that if the starting coordinates are different, then the two obtained sequence must be different. Suppose \(y_1\) and \(y_2\) are two sequences in \(B'_{k,\ell}(x)\), and their errors start at the \(i_1\)th and \(i_2\)th coordinate of \(x\), respectively (WLOG, assume \(i_1 < i_2\)). Then \(y_1\) and \(y_2\) have distinct symbols on their \(i_1\)th coordinate and thus \(y_{i_1} \neq y_{i_2}\). To calculate \(|B'_{k,\ell}(x)|\), we have \(n-k+1\) choices for \(i\), and once \(i\) is given the inserted sequence has \(2^{\ell-2}\) possibilities since \(y_{i_1} = 1 - x_i\) and \(y_{i_2} = 1 - x_{i+k-1}\) are fixed. As a consequence, there are altogether \((n - k + 1) \cdot 2^{\ell-2}\) distinct sequences in \(B'_{k,\ell}(x)\).

Lemma 4: Let \(x \in \mathbb{F}_2^n\), \(k = 0\), \(\ell \geq 1\), then
\[
|B'_{0,\ell}(x)| = n \cdot 2^{\ell-1} + 2^{\ell}.
\]

Proof: For any \(y \in B'_{0,\ell}(x)\), let \(1 \leq i \leq n\) be the first coordinate where \(x\) and \(y\) differ, then \(y\) can be seen as inserting a sequence of length \(\ell\) at the \(i\)th coordinate and there are \(2^{\ell-1}\) possibilities for the inserted sequence. If no such \(i\) exists, which means the first \(n\) symbols of \(y\) equal \(x\), then \(y\) can be seen as inserting a sequence of length \(\ell\) at the end of \(x\) and there are \(2^\ell\) possibilities for the inserted sequence.

To sum up we have \(|B'_{0,\ell}(x)| = n \cdot 2^{\ell-1} + 2^\ell\).

Summarizing the previous lemmas, the size of \(|B'_{k,\ell}(x)|\) is as follows:
\[
B'_{k,\ell}(x) = \begin{cases}
    1 + \sum_{i=1}^{k} (r(A_k(x)_i) - 1), & \text{if } k \geq 0, \ell = 0 \\
    n - \sum_{i=1}^{k} r(A_{k-1}(x)_i), & \text{if } k \geq 1, \ell = 1 \\
    (n - k + 1) \cdot 2^{\ell-2}, & \text{if } k \geq 1, \ell \geq 2 \\
    n \cdot 2^{\ell-1} + 2^\ell, & \text{if } k = 0, \ell \geq 1
\end{cases}
\]

Now we are ready to compute the size of the error ball \(B_{t,s}(x)\).

Theorem 3: For \(x \in \{0,1\}^n\), \(t \geq 1\), \(s \geq 1\), the size of a \((t,s)\)-burst error ball is
\[
|B_{t,s}(x)| = (n - t + 2) \cdot 2^{s-1}.
\]

Proof: If \(t \geq s\), from Lemma 1, we can calculate
\[
|B_{t,s}(x)| = \sum_{\ell=0}^{t-s} |B'_{s-k,\ell}(x) |
\]
\[
= 1 + \sum_{i=1}^{t-s} (r(A_{t-s}(x)_i) - 1) + n - \sum_{i=1}^{t-s} r(A_{t-s}(x)_i)
\]
\[
= \sum_{\ell=2}^{s} (n - t + s - \ell - 1) \cdot 2^{\ell-2}
\]
\[
= n + s - t + 1 + \sum_{\ell=0}^{s-2} (n - t + s - \ell - 1) \cdot 2^{\ell}
\]

Let \(S = \sum_{t=0}^{s-2} (n - t + s - \ell - 1) \cdot 2^{\ell}\), then \(2S = \sum_{t=0}^{s-2} (n - t + s - \ell - 1) \cdot 2^{\ell+1} = \sum_{\ell=0}^{s-1} (n - t + s - \ell) \cdot 2^\ell\). So, we can
calculate $S$ as

$$S = 2S - S = \sum_{\ell=1}^{s-1} (n - t + s - \ell) \cdot 2^\ell - \sum_{\ell=0}^{s-2} (n - t + s - \ell - 1) \cdot 2^\ell = (n - t + 1) \cdot 2^{s-1} + \sum_{\ell=1}^{s-2} 2^\ell - (n - t + s - 1) = (n - t + 2) \cdot 2^{s-1} - n + t - s - 1.$$ 

Thus, the size of the error ball is

$$|B_{t,s}(x)| = n + s - t + 1 + \sum_{\ell=0}^{s-2} (n - t + s - i - 1) \cdot 2^\ell = n + s - t + 1 + (n - t + 2) \cdot 2^{s-1} - n + t - s - 1 = (n - t + 2) \cdot 2^{s-1}.$$

Similarly, if $t < s$, from Lemma 1, we can calculate

$$|B_{t,s}(x)| = n \cdot 2^{s-t-1} + 2^{s-t} + \sum_{k=1}^{t} (n - k + 1) \cdot 2^{s-t+k-2} = (n - t + 2) \cdot 2^{s-1}.$$ 

Summing up the above, we have proved that for any $t \geq 1, s \geq 1$, $|B_{t,s}(x)| = (n + t + 2) \cdot 2^{s-1}$.

**Example 2:** Let $n = 12, x = 101011100100, t = 4, s = 1$, then

$$A_3(x) = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}.$$ 

$$B'_{3,0}(x) = \{011100100, 111100100, 101100100, 100100100, 100101000, 100111000\},$$

$$B'_{4,1}(x) = \{1001000100, 101011000, 101011110, 101011101\}.$$ 

Hence, $|B'_{3,0}(x)| + |B'_{4,1}(x)| = 10 = (12 - 4 + 2) \cdot 2^{4-1}$.

**Theorem 4:** Let $C \subseteq \mathbb{F}_2^n$ be a $(t, s)$-burst correcting code where $t, s \geq 1$, then

$$|C| \leq \frac{2n-t+1}{n-t+2}.$$ 

**Proof:** Given any two words $c_1, c_2 \in C$,

$$B_{t,s}(c_1) \cap B_{t,s}(c_2) = \emptyset.$$ 

Now consider the union of all $(t, s)$-burst balls centered at the codewords in $C$. Obviously, their union is a subset of $\mathbb{F}_2^{n-t+s}$.

In other words,

$$\bigcup_{c \in C} B_{t,s}(c) \subseteq \mathbb{F}_2^{n-t+s}.$$ 

Since any two $(t, s)$-burst balls centered at distinct codewords in $C$ are disjoint,

$$\left| \bigcup_{c \in C} B_{t,s}(c) \right| = \sum_{c \in C} |B_{t,s}(c)| = (n - t + 2) \cdot 2^{s-1} \cdot |C|.$$ 

Consequently,

$$|C| \leq \frac{2n-t+1}{n-t+2}.$$ 

Note that for $t = s = 1$ the upper bound above coincides with the Hamming bound for a single substitution, and when $t = s$ our upper bound coincides with the upper bound of burst-error correcting code for substitutions derived by Dass [3]. Moreover, it is quite interesting that the upper bound above is irrelevant to the number of inserted symbols $s$. Recall the equivalence of $(t, s)$-burst correcting codes and $(s, t)$-burst correcting codes, $t$ can be replaced by $s$ in the upper bound of Theorem 4, and the better bound is the one using the relatively larger value between $t$ and $s$. Therefore, from now on we only focus on the case $t \geq s$, and the redundancy of a $(t, s)$-burst correcting code is lower bounded by

$$n - \log |C| \geq \log(n + t + 2) + t - 1.$$ 

We close this section by discussing the redundancy of $(\leq t, \leq s)$-burst correcting codes. By Corollary 1 we only consider the case when $t \geq s$. It is easy to see that a $(t, s)$-burst correcting code can also correct a $(t - e, s - e)$-burst error for any integer $0 \leq e \leq s$, as we can delete an extra subsequence of length $e$ followed by the deleted $t - e$ symbols and then append this length $e$ subsequence after the inserted $s - e$ symbols. Therefore, any $(t', s')$-burst error can be viewed as a $(t, s' + t - t')$-burst error for $t' \leq t$. This can be summarized as the following theorem.

**Theorem 5:** For $t \geq s$, a code is a $(\leq t, \leq s)$-burst correcting code if and only if it can correct burst errors of types $\{(t, 0), (t, 1), \ldots, (t, s), (t - 1, s), \ldots, (s, s)\}$.

The theorem indicates that the size of a $(\leq t, \leq s)$-burst correcting code is naturally upper bounded by the corresponding sphere-packing upper bound for each $(t', s')$-burst correcting code, where $(t', s') \in \{(t, 0), (t, 1), \ldots, (t, s), (t - 1, s), \ldots, (s, s)\}$. Note that these upper bounds can be easily derived as shown in Theorem 4, and the only exception is the upper bound for $t$-burst-deletion-correcting codes (so far the best one is from [19]), as we have mentioned in the start of
this section). Therefore, for a \((\leq t, \leq s)\)-burst correcting code \(C\), we can get

\[
|C| \leq \min_{s \leq t' \leq t} \left\{ \frac{2^{n-t+1}}{n-t'+2} \cdot \frac{2^{n-t+1} - 2^t}{n-2t+1} \right\} = \frac{2^{n-t+1}}{n-t+2},
\]

which means the redundancy is also at least \(\log(n-t+2)+t-1\). However, we note that this upper bound may be quite weak since we do not take into consideration that the code should be able to correct all \((t', s')\)-burst errors simultaneously, where \((t', s') \in \{(t,0), (t,1), \ldots, (t,s), (t-1,s), \ldots, (s,s)\}\). The redundancy for \((\leq t, \leq s)\)-burst correcting codes is left as an open problem.

V. A GENERAL CONSTRUCTION OF \((t, s)\)-BURST CORRECTING CODES FOR \(t \geq 2s\)

The equivalence allows us to focus only on the case \(t \geq s\), i.e., the number of deletions is no less than the number of insertions. In this section, we provide a construction of \((t, s)\)-burst correcting codes for \(t \geq 2s\). We start with an overall explanation of the main idea, and then proceed with the detailed construction and analysis followed by some discussions.

A. Sketch of the Main Idea

In the array representation \(A_{t-s}(x)\), if \(t \geq 2s\), then it is routine to check that a \((t, s)\)-burst results in an array of size \((t-s) \times (\frac{t}{s}-1)\) and each row of \(A_{t-s}(x)\) suffers from either one deletion or a \((2, 1)\)-burst. To be more specific, suppose the starting coordinate of the error is located at the \(k\)th row of \(A_{t-s}(x)\), then each of the rows indexed by \(k, k+1, \ldots, k+s-1\) (indices are calculated modulo \(t-s\)) suffers from a \((2,1)\)-burst and each of the rest rows suffers from a single deletion. The following toy example help visualize the error type on each row. Let \(t = 5\), \(s = 2\), and \(n = 12\). Below are the error patterns if the error starts on the 3rd, 4th, or the 5th coordinate, correspondingly.

\[
\begin{align*}
& x_1 \ x_4 \ x_7 \ x_{10} \\
& x_2 \ x_5 \ x_8 \ x_{11} \\
& x_3 \ x_6 \ x_9 \ x_{12}
\end{align*} \quad \rightarrow \quad
\begin{align*}
& x_1 \ y_2 \ x_{10} \\
& x_2 \ y_3 \ x_{11} \\
& y_4 \ x_9 \ x_{12},
\end{align*}
\]

\[
\begin{align*}
& x_1 \ x_4 \ x_7 \ x_{10} \\
& x_2 \ x_5 \ x_8 \ x_{11} \\
& x_3 \ x_6 \ x_9 \ x_{12}
\end{align*} \quad \rightarrow \quad
\begin{align*}
& x_1 \ y_2 \ x_{10} \\
& x_2 \ y_3 \ x_{11} \\
& y_4 \ x_9 \ x_{12},
\end{align*}
\]

\[
\begin{align*}
& x_1 \ x_4 \ x_7 \ x_{10} \\
& x_2 \ x_5 \ x_8 \ x_{11} \\
& x_3 \ x_6 \ x_9 \ x_{12}
\end{align*} \quad \rightarrow \quad
\begin{align*}
& x_1 \ x_4 \ y_{10} \\
& x_2 \ y_3 \ x_{11} \\
& y_4 \ x_9 \ x_{12},
\end{align*}
\]

Recall that the \((2,1)\)-burst correcting code as shown in Equation (2) can also correct a single deletion, due to the VT syndrome in its definition. We can let the first row of \(A_{t-s}(x)\) be chosen from a \((2,1)\)-burst correcting code, and decoding this row will provide us some additional knowledge about the location of the error in the other rows. To be more specific, the decoding of the first row (using a decoder for a \((2,1)\)-burst correcting code) falls into three cases.

1. The first row suffers from a \((2,1)\)-burst where the error is of the form \(00 \rightarrow 1\) or \(11 \rightarrow 0\).
2. The first row suffers from a \((2,1)\)-burst which does not belong to Case 1.
3. The first row suffers from a single deletion.

For Case 1, we can uniquely determine the two erroneous coordinates in the first row. Since \(t \leq 2(t-s)\), the deleted coordinates are within at most three consecutive columns. Therefore, the erroneous coordinates in the other rows are located within an interval of length at most 3. See the previous example, if we have determined that the first row suffers from a \((2,1)\)-burst at the second coordinate (the first one and the middle one), then the erroneous coordinates on the other rows are only of the following possibilities: a \((2,1)\)-burst at the first coordinate (3rd row in the first one), a \((2,1)\)-burst at the second coordinate (2nd row in the middle one), or a single deletion at the second coordinate.

Case 2 can be also seen as a single deletion in the first row. Therefore we can combine Case 2 and Case 3 together, and locate the single deletion in the first row within a run, which might range from the \(c_1\)th to the \(c_2\)th column, for some \(c_1 \leq c_2\). Then the errors in the other rows should be within columns indexed from \(c_1-1\) to \(c_2\). By adding additional run-length constraint on the first row, \(c_2 - c_1\) is bounded and then we may apply a \((2,1)\)-burst-SVT code (to be defined later), which can correct a \((2,1)\)-burst with the additional knowledge about the location of the error within an interval of \(P\) consecutive coordinates.

Here is an example to illustrate the whole decoding process.

**Example 3:** Let \(x = 101011001101110\), \(t = 4, s = 1\), and the erroneous sequence is \(x' = 1010101101110\). First write \(x'\) in the form of an array,

\[
A_3(x') = \begin{bmatrix}
1 & 0 & 1 & 1 \\
0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0
\end{bmatrix}.
\]

Applying the decoder of a \((2,1)\)-burst correcting code to the first row, we get 10011. In addition we know that the first row suffers from a single deletion on either the second or the third coordinate. Therefore, the errors in the remaining two rows are within their first three coordinates. Applying the decoder of a \((2,1)\)-burst SVT code with \(P = 3\) to them, we get the second and third rows as 01001, 11110 respectively. Consequently,

\[
A_3(x) = \begin{bmatrix}
1 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 \\
1 & 1 & 1 & 1 & 0
\end{bmatrix},
\]

and \(x = 1010110011011110\) is correctly decoded.

We close this subsection by discussing the parameter \(P\) in the run-length constraint [13]. If \(P\) is too large (e.g. linear in \(n\)), using \((2,1)\)-burst SVT codes does not reduce the total redundancy as compared to just using \((2,1)\)-burst correcting codes. Therefore, to further reduce the redundancy we want to make \(P\) small. Define \(S_n(f(n))\) to be the set of binary sequences of length \(n\) whose maximum run length is at most \(f(n)\). An algorithm is provided to efficiently encode any binary sequence to a \((\log n + 3)\)-RLL sequence with only 1 bit of redundancy in [19], i.e., \(|S_n(\log n + 3)| \geq 2^n-1\). Once we use a subcode of \(S_n(\log n + 3)\) in the first row, it is guaranteed...
that for the other rows we have the additional knowledge about the location of the error within a bounded interval.

Now, we are fully prepared to present our main construction.

B. A General Construction for \( t \geq 2s \)

We follow the array representation framework to write \( x \) as an array \( A_{t-s}(x) \) of size \((t-s) \times \frac{n}{t-s}\). The first row \( A_{t-s}(x) \) comes from the following code.

**Construction 1:** For arbitrary integers \( n \) and \( a \in \mathbb{Z}_{2n-1} \), \( b \in \mathbb{Z}_4 \), define \( c_{2,1}^{RL}(n; a, b, \log n + 3) \) as

\[
C_{2,1}^{RL}(n; a, b, \log n + 3) = C_{2,1}(n; a, b) \cap S_n(\log n + 3).
\]

**Theorem 6:** For arbitrary integers \( n \), there exists a code \( C_{2,1}^{RL}(n; a, b, \log n + 3) \) with redundancy at most \( \log n + 4 \).

**Proof:** Recall that \( |S_n(\log n + 3)| \geq 2^{n-1} \). Moreover, for \( a \in \mathbb{Z}_{2n-1} \), \( b \in \mathbb{Z}_4 \), \( \sum_{i,j} c_{2,1}(n; a, b) \cap S_n(\log n + 3) \) is a disjoint partition of \( S_n(\log n + 3) \). Thus according to the pigeonhole principle, there must exist choices for \( a \in \mathbb{Z}_{2n-1} \) and \( b \in \mathbb{Z}_4 \), such that

\[
|c_{2,1}^{RL}(n; a, b, \log n + 3)| \geq \frac{2^n}{4(2n - 1)}.
\]

Therefore, the redundancy is at most

\[
n - \log |C_{2,1}^{RL}(n; a, b, \log n + 3)| = \log 4(2n - 1) + 1 < \log n + 4.
\]

Due to the RLL constraint of the first row, the starting coordinate of the error on each remaining row will be limited to an interval of length \( \log n + 4 \). Now, we will provide a code which can correct a \((2,1)\)-burst with this additional knowledge.

**Construction 2:** For arbitrary integers \( n \) and \( c \in \mathbb{Z}_{2^p-1} \), \( d \in \mathbb{Z}_4 \), \( P \leq n \), define the \((2,1)\)-burst-SVT code as

\[
SVT_{2,1}^{burst}(n; c, d, P) = \left\{ x : VT(x) \equiv c \pmod{2^p - 1}, \sum_{i=1}^n x_i \equiv d \pmod{4} \right\}.
\]

**Theorem 7:** The code \( SVT_{2,1}^{burst}(n; c, d, P) \) can correct a \((2,1)\)-burst with the additional knowledge of the starting location of the \((2,1)\)-burst within an interval of \( P \) consecutive coordinates. Furthermore, there exist choices for \( c \) and \( d \) such that the redundancy of the code is at most \( \log P + 3 \).

**Proof:** For any \( u \in \mathbb{P}_2^n \) that suffers from a \((2,1)\)-burst, denote the received sequence as \( u' \in \mathbb{P}_2^{n-1} \). Define \( \Delta = \sum_{i=1}^n u_i - \sum_{i=1}^{n-1} u'_i \pmod{4} \), \( \Delta \in \{0, 1, 2, 3\} \). The decoding starts with the observation of the value \( \Delta \).

We have \( \Delta = 3 \) if and only if \( 00 \rightarrow 1 \) happens. Similarly, \( \Delta = 2 \) if and only if \( 11 \rightarrow 0 \) happens. For the remaining cases, when \( \Delta = 0 \) or \( 1 \), the \((2,1)\)-burst error could be seen as just a single deletion. Since \( SVT_{2,1}^{burst}(n; c, d, P) \) is also an SVT code (due to the VT syndrome in its definition), it can correct a single deletion with the additional knowledge of the deleted coordinate within a length-\( P \) interval.

We are only left with the case when the error is \( 11 \rightarrow 0 \) or \( 00 \rightarrow 1 \). We only prove the former case \( 11 \rightarrow 0 \). The latter case \( 00 \rightarrow 1 \) can be proved analogously.

Suppose there exist \( x, y \in SVT_{2,1}^{burst}(n; c, d, P) \), such that \( z \in B_{2,1}(x) \cap B_{2,1}(y) \). Suppose the \((2,1)\)-burst of the form \( 11 \rightarrow 0 \) starts at \( i \)th coordinate of \( x \) and \( j \)th coordinate of \( y \), and without loss of generality \( i < j \). We have

\[
x = (x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_j, 0, x_{j+1}, \ldots, x_n),
\]

\[
y = (y_1, \ldots, y_{i-1}, 0, y_{i+1}, \ldots, y_j, 1, y_{j+1}, \ldots, y_n),
\]

where \( x_k = y_k \) when \( 1 \leq k \leq i - 1 \) and \( j + 2 \leq k \leq n \), \( x_{k+1} = y_k \) when \( i + 1 \leq k \leq j - 1 \).

Now we consider the difference of syndromes \( VT(x) - VT(y) \), which is equal to

\[
\sum_{i=1}^n ix_i - \sum_{i=1}^n iy_i = (i + 1) + wt(x_{i+2}, \ldots, x_j) - j - (j + 1) = 2(i - j) + wt(x_{i+2}, \ldots, x_j),
\]

where \( wt(u) \) is the Hamming weight of \( u \). Since \( \leq wt(x_{i+2}, \ldots, x_j) \leq j - i - 1 \), we have

\[
2(i - j) \leq VT(x) - VT(y) \leq i - j - 1.
\]

Furthermore, \( i \) and \( j \) are within an interval of length \( P \), so \( 1 \leq j - i \leq P - 1 \). Hence,

\[
-2(P - 1) \leq VT(x) - VT(y) \leq -2.
\]

Therefore, \( VT(x) - VT(y) \neq 0 \pmod{2^P - 1} \), which contradicts the fact that \( x, y \in SVT_{2,1}^{burst}(n; c, d, P) \). Thus \( SVT_{2,1}^{burst}(n; c, d, P) \) can uniquely correct a \((2,1)\)-burst error with the additional knowledge of the starting location of the \((2,1)\)-burst within an interval of \( P \) consecutive coordinates.

Moreover, since \( \bigcup_{c,d} SVT_{2,1}^{burst}(n; c, d, P) \) is a partition of \( \mathbb{P}_2^n \), according to the pigeonhole principle, there must exist \( c \) and \( d \) such that the code size is at least \( \frac{2^n}{4(2^p - 1)} \), thus the redundancy of the code is at most \( \log(2^P + 1) < \log P + 3 \).

Now, we are ready to present our construction of \((t, s)\)-burst correcting codes for \( t \geq 2s \).

**Construction 3:** Let \( t, s \geq 1 \) such that \( t \geq 2s \). Let \( a \in \mathbb{Z}_{2n/(t-s)-1} \), \( b \in \mathbb{Z}_4 \), and \( c_i \in \mathbb{Z}_{2^p-1} \), \( d_i \in \mathbb{Z}_4 \), where \( 2 \leq i \leq t - s \), \( P = \frac{n}{t-s} + 4 \). The code \( C_{t,s} \) is constructed as follows:

\[
C_{t,s} = \left\{ x : A_{t-s}(x) \in C_{2,1}^{RL}(n; a, b, \log n + 3), A_{t-s}(x) \in SVT_{2,1}^{burst}(n; \frac{n}{t-s}; c_i, d_i, \log n + 4), \right. \]

\[
\left. 2 \leq i \leq t - s \right\}.
\]

**Theorem 8:** For \( t \geq 2s \), the code \( C_{t,s} \) is a \((t, s)\)-burst correcting code, and there exist choices for \( a, b, c_i, d_i \) such that the redundancy of the code is at most \( \log((t-s+1) \log n + O(1)).

**Proof:** Suppose \( x \in C_{t,s} \) suffers from a \((t, s)\)-burst and the erroneous sequence is denoted as \( y \). Consider them in the array form \( A_{t-s}(x) \) and \( A_{t-s}(y) \). The \((t, s)\)-burst will cause either a \((2,1)\)-burst or a single deletion in the first row, which could be correctly decoded since the first row comes from
a (2,1)-burst correcting code. After decoding $A_{t-s}(x)_1$ and comparing with $A_{t-s}(y)_1$, due to the run-length constraint of the first row, we will get some additional knowledge.

On one hand, if the first row suffers from a single deletion, then we may locate its deleted coordinate within an interval of length $\log \frac{n}{t-s} + 3$. The starting coordinate of error in the other rows will be located within an interval of length $\log \frac{n}{t-s} + 4$. Then the rest rows can be uniquely decoded due to the property of the (2,1)-burst SVT code.

On the other hand, if the first row suffers from a (2,1)-burst in the form of $00 \rightarrow 1$ or $11 \rightarrow 0$, then we may locate the erroneous coordinates on the first row and the erroneous coordinates on the other rows are within an interval of length 3, thus again the rest rows can be uniquely decoded due to the property of the (2,1)-burst SVT code.

To sum up, $C_{t,s}$ is indeed a $(t,s)$-burst correcting code.

As for the size of the code, by the pigeonhole principle there must exist choices for $a, b, c, d, 2 \leq i \leq t - s$, such that

$$|C_{t,s}| \geq \frac{2^{n/2} - 1 \cdot (2^{n/2})^{t-s}}{4 \cdot (2^{n/2} - 1) \cdot (4 \cdot (2 \log \frac{n}{t-s} + 4) - 1)}^{t-s-1}.$$  

Hence, the redundancy is at most

$$4 + \log \frac{n}{t-s} + (t-s-1)(\log \log \frac{n}{t-s} + 4) + 3),$$

which is $\log n + (t-s-1) \log \log n + O_t(1)$.

C. Further Discussions

Note that, $(t,s)$-burst correcting codes can correct $(t-s,0)$-burst errors (i.e., $(t-s)$-burst-deletion), then $(t,s)$-burst correcting codes are naturally also $(t-s)$-burst-deletion-correcting codes. Regarding the redundancy, the optimal redundancy of $(t,s)$-burst correcting codes should be lower bounded by the optimal redundancy of $(t-s)$-burst-deletion-correcting codes.

Up till now the best known construction of $(t-s)$-burst-deletion-correcting codes is the one from [19], where the redundancy is about $\log n + (t-s-1) \log \log n + O_t(1)$. Comparing with the redundancy of our $(t,s)$-burst correcting codes from Construction 3, there is only a difference of a constant term. Therefore, as a byproduct, our codes from Construction 3 also performs well against only $(t-s)$-burst-deletions.

Furthermore, note that there is only a $\log \log n$ gap between the redundancy of Construction 3 and the lower bound suggested by the sphere-packing bound. In the next section, for $t=3$ and $s=1$ we manage to close this gap.

VI. Optimal Codes Correcting A (3,1)-Burst

A (2,1)-burst error, except for the two forms $11 \rightarrow 0$, $00 \rightarrow 1$, can be seen as a single deletion. The almost optimal $(2,1)$-burst correcting code with redundancy $\log n + 3$ from [19], as shown in Equation (2), adds an additional constraint on the basis of the VT code, in order to deal with the errors of the form $11 \rightarrow 0$ and $00 \rightarrow 1$. Motivated by this construction and the fact that a large proportion of (3,1)-burst errors could be seen as a 2-burst-deletion, we build our code based on Levenshtein’s code for 2-burst-deletions as shown in Equation (1), and add some additional constraints in order to deal with the other kinds of (3,1)-burst errors.

Construction 4: For $a \in \mathbb{Z}_{4n}$, $b, c \in \mathbb{Z}_4$, and $d \in \mathbb{Z}_5$, the code $C_{3,1}(n; a, b, c, d)$ is defined as follows:

$$C_{3,1}(n; a, b, c, d) = \{ x : \text{Rsgn}(0x) \equiv a \pmod{4n}, \sum_{i=1}^{n/2} x_{2i-1} \equiv b \pmod{4}, \sum_{i=1}^{n/2} x_{2i} \equiv c \pmod{4}, r(x) \equiv d \pmod{5} \}.$$  

In order to show that $C_{3,1}(n; a, b, c, d)$ is a (3,1)-burst correcting code we need several steps. First we divide all (3,1)-burst errors into two types. The first type contains those which can be seen as a 2-burst-deletion, for example, $001 \rightarrow 1$, $000 \rightarrow 0$. The second type only consists of the following: $\{000 \rightarrow 1, 010 \rightarrow 1, 111 \rightarrow 0, 101 \rightarrow 0 \}$. To verify which type of error occurs, we need to observe the following two values. Suppose $u \in \mathbb{F}_2^n$ is a codeword in $C_{3,1}(n; a, b, c, d)$, and $u' \in \mathbb{F}_{2^2}-2$ is the sequence obtained from $u$ by a (3,1)-burst. Let $\Delta_{odd}(u) = \sum_{i=1}^{n/2} u_{2i-1} - \sum_{i=1}^{n/2-1} u'_{2i-1}$ (mod 4) and $\Delta_{even}(u) = \sum_{i=1}^{n/2} u_{2i-1} - \sum_{i=1}^{n/2-1} u'_{2i-1}$ (mod 4).

Lemma 5: By observing $\Delta_{odd}(u)$ and $\Delta_{even}(u)$, one can verify whether the (3,1)-burst error is a 2-burst-deletion or is of the form $\{000 \rightarrow 1, 010 \rightarrow 1, 111 \rightarrow 0, 101 \rightarrow 0 \}$.

Proof: Suppose the (3,1)-burst error starts at the $i$th coordinate of $u = (u_1, u_2, \ldots, u_n)$, and results in $(u_1, \ldots, u_{i-1}, y, u_{i+2}, \ldots, u_n)$, where $y$ is the inserted symbol. If $i$ is odd, then $\Delta_{odd}(u) = u_i + u_{i+2} - y$ and $\Delta_{even}(u) = u_i+1$. Similarly, if $i$ is even then $\Delta_{even}(u) = u_i + u_{i+2} - y$ and $\Delta_{odd}(u) = u_{i+1}$. The correspondence of the error type and $(\Delta_{odd}(u), \Delta_{even}(u))$ is as follows:

- $000 \rightarrow 1$. Then $(\Delta_{odd}(u), \Delta_{even}(u))$ is either $(3,0)$ or $(0,3)$.
- $010 \rightarrow 1$. Then $(\Delta_{odd}(u), \Delta_{even}(u))$ is either $(3,1)$ or $(1,3)$.
- $111 \rightarrow 0$. Then $(\Delta_{odd}(u), \Delta_{even}(u))$ is either $(2,1)$ or $(1,2)$.
- $101 \rightarrow 0$. Then $(\Delta_{odd}(u), \Delta_{even}(u))$ is either $(2,0)$ or $(2,0)$.

If the error is a 2-burst-deletion, then the two deleted symbols may be $00, 01, 10, 11$ and thus $(\Delta_{odd}(u), \Delta_{even}(u)) \in \{(0,0), (0,1), (1,0), (1,1)\}$.

Thus, the disjointness of the possible values of $(\Delta_{odd}(u), \Delta_{even}(u))$ allows us to precisely determine the error type.

The first step of our decoding process is to observe $(\Delta_{odd}(x), \Delta_{even}(x))$ based on Lemma 5. Suppose that we have verified that the error is indeed a 2-burst-deletion. Then, as proved by Levenshtein [16], the constraint $\text{Rsgn}(0x) \equiv a \pmod{4n}$ in the code $C_{3,1}(n; a, b, c, d)$ guarantees that we can successfully decode any 2-burst-deletion error. Therefore, it suffices to analyze the case when the error is of the form $\{000 \rightarrow 1, 010 \rightarrow 1, 111 \rightarrow 0, 101 \rightarrow 0 \}$.

In the rest of this section we only prove that the code $C_{3,1}(n; a, b, c, d)$ is an error-correcting code against the error type $000 \rightarrow 1$ or the error type $010 \rightarrow 1$. It should be noted in advance that the proofs of the other two error types are
not simply modified by flipping 0’s and 1’s, since they are in fact not completely symmetrical due to the ‘0’ in $R_{syn}(0x)$.

However, they can be proved by a similar (and even easier) deduction as the following two lemmas. Due to the strong similarity we choose to omit the redundant proofs and leave the remaining two error types to the reader. For convenience, define $x_{[i,j]} = x_i \ldots x_j$ as the substring of $x$ which starts at the $i$th coordinate and ends at the $j$th coordinate.

**Lemma 6:** $C_{3,1}(n; a, b, c, d)$ is an error-correcting code against a $000 \rightarrow 1$ error.

**Proof:** Suppose we have two distinct codewords $x, y \in C_{3,1}(n; a, b, c, d)$ and $z \in \mathbb{F}_2^{n-2}$ can be derived from both $x$ and $y$ by a $000 \rightarrow 1$ error. Then $x$ and $y$ should be of the form

$$x = (x_{[i, i-1]}, 0, 0, 0, x_{[i+3, j+1]}, 1, x_{[j+3, n]}),$$

$$y = (y_{[i, i-1]}, 1, y_{[i+1, j+1]}, 0, 0, y_{[j+3, n]}),$$

where $x_k = y_k$ when $1 \leq k \leq i - 1$ and $j + 3 \leq k \leq n$, $x_{k+2} = y_k$ when $i + 1 \leq k \leq j - 1$.

We turn to the last constraint $r(x)$ in the definition of $C_{3,1}(n; a, b, c, d)$, which is the number of runs. Let $\Delta_r(x) = r(x) - r(z)$ (mod 5) and $\Delta_r(y) = r(y) - r(z)$ (mod 5).

Since $x, y$ are both codewords from $C_{3,1}(n; a, b, c, d)$, we must have $\Delta_r(x) = \Delta_r(y)$ and its value is of the following possibilities:

- If $i = 1$, i.e., the $(3, 1)$-burst starts at the very beginning, then depending on whether $x_0 = 0$ or $x_4 = 1$, the error pattern is either $0000 \rightarrow 10$ or $0001 \rightarrow 11$ and thus $\Delta_r(x) = 4$ or 1.

- If $i = n - 2$, i.e., the $(3, 1)$-burst starts at the end, then depending on whether $x_{n-3} = 0$ or $x_{n-3} = 1$, the error pattern is either $0000 \rightarrow 01$ or $1000 \rightarrow 11$ and thus $\Delta_r(x) = 4$ or 1.

- Otherwise, consider the four different cases for $x_{i-1}$ and $x_{i+3}$. The error pattern is $00000 \rightarrow 0100001 \rightarrow 011100001 \rightarrow 110100001 \rightarrow 111$ and the corresponding value of $\Delta_r(x)$ is 3, 0, 0, and 2.

The rest of the proof falls into five cases. In each case we arrive at a contradiction and thus prove that the assumption of the two distinct codewords $x, y \in C_{3,1}(n; a, b, c, d)$ is not valid.

- Case 1: $\Delta_r(x) = \Delta_r(y) = 1$. Then according to the analysis above, we have

$$x = (0, 0, 0, 0, x_{[5, n-2]}, 1, 1),$$

$$y = (1, 1, y_{[3, n-4]}, 1, 0, 0, 0),$$

where $x_{k+2} = y_k$ when $3 \leq k \leq n - 4$ and the 000 $\rightarrow 1$ error starts at the first coordinate of $x$ and the $(n-2)$th coordinate of $y$. Now we turn back to the constraint $R_{syn}(0x)$ for contradiction. The run sequence $R(0x)$ is of the form

$$0, 0, 0, 0, 1, R(0x)_{[6, n-1]}, \lambda - 1, \lambda - 1,$$

where $\lambda = r(0x) = r(x)$. The run sequence $R(0y)$ is of the form

$$0, 1, 1, R(0y)_{[4, n-3]}, \lambda' - 2, \lambda' - 1, \lambda' - 1, \lambda' - 1,$$

where $\lambda' = r(0y) = r(y) + 1$. Note that for every $3 \leq k \leq n - 4$, the run index of $x_{k+2}$ in $R(0x)$ equals the run index of $y_k$ in $R(0y)$. Also note that according to the definition of the code we must have $r(x) = r(y)$. Thus we have

$$R_{syn}(0x) - R_{syn}(0y) = (2\lambda - 1) - (4\lambda' - 3) = -2r(x) - 2 \neq 0 \pmod{4n},$$

which is a contradiction to the constraint $R_{syn}(0x) \equiv R_{syn}(0y) \pmod{4n}$.

- Case 2: $\Delta_r(x) = \Delta_r(y) = 4$. Similarly as the previous case, we have

$$x = (0, 0, 0, 0, x_{[5, n-2]}, 0, 1),$$

$$y = (1, 0, y_{[3, n-4]}, 0, 0, 0, 0),$$

where $x_{k+2} = y_k$ when $3 \leq k \leq n - 4$ and the 000 $\rightarrow 1$ error starts at the first coordinate of $x$ and the $(n-2)$th coordinate of $y$. The run sequence $R(0x)$ is of the form

$$(0, 0, 0, 0, 0, R(0x)_{[6, n-1]}, \lambda - 2, \lambda - 1),$$

where $\lambda = r(0x) = r(x)$. The run sequence $R(0y)$ is of the form

$$(0, 1, 2, R(0y)_{[4, n-3]}, \lambda' - 1, \lambda' - 1, \lambda' - 1, \lambda' - 1),$$

where $\lambda' = r(0y) = r(y) + 1$. Note that for every $3 \leq k \leq n - 4$, the run index of $x_{k+2}$ in $R(0x)$ is the run index of $y_k$ in $R(0y)$ minus two. Thus we have

$$R_{syn}(0x) - R_{syn}(0y) = (2\lambda - 3) - 2(n - 6) - (4\lambda' - 1) = -2n - 2r(x) + 6 \pmod{4n}.$$ 

From the representation of $x$, we have $2 \leq r(x) \leq n - 3$. Thus $-4n + 12 \leq -2n - 2r(x) + 6 \leq -2n + 2$ and thus $R_{syn}(0x) - R_{syn}(0y)$ is nonzero modulo 4n, a contradiction.

- Case 3: $\Delta_r(x) = \Delta_r(y) = 3$ and the error pattern must be 00000 $\rightarrow 010$. Hence $x$ and $y$ are

$$(\ldots, 0, 0, 0, 0, 0, x_{[i+4, j]}, 0, 1, 0, \ldots),$$

$$(\ldots, 0, 1, a, a, a, a, b, b+1, b+2, \ldots),$$

$$0, 0, \ldots, 0, y_{[i+2, j-2]}, b+2, b+2, b+2, b+2, b+2, b+2, \ldots,$$

where the $a$'s and $b$'s are the run indices of corresponding entries. Moreover, for every $i + 2 \leq k \leq j - 2$, the run index of $x_{k+2}$ in $R(0x)$ is the run index of $y_k$ in $R(0y)$ minus two. In this case,

$$R_{syn}(0x) - R_{syn}(0y) = (5a + 3b + 3) - 2(j - i - 3) - (3a + 5b + 13) = 2(a - b) - 2(j - i) - 4 \pmod{4n}.$$ 

From the representation of $x$ we have $a \leq b \leq a + j - i - 2$. Hence, $-4(j - i) \leq R_{syn}(0x) - R_{syn}(0y) \leq -2(j - i) - 4$. Moreover, $2 \leq j - i \leq n - 5$, thus we have $-4(n - 5) \leq R_{syn}(0x) - R_{syn}(0y) \leq -8$, which is again a contradiction to the constraint that $R_{syn}(0x) - R_{syn}(0y) \equiv 0 \pmod{4n}$. 

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
• Case 4: $\Delta_r(x) = \Delta_r(y) = 2$ and the error pattern must be $1001 \rightarrow 111$. We have

$$x = (\ldots, 0, 0, 0, 0, 1, x_{[i+4,j]}, 1, 1, \ldots),$$

$$y = (\ldots, 1, 1, y_{[i+2,j-2]}, 1, 0, 0, 0, 0, 1, \ldots).$$

In this case, for every $i + 2 \leq k \leq j - 2$, the run index of $x_{k+2}$ in $R(0x)$ is the run index of $y_k$ in $R(0y)$ plus two. Then we have

$$Rsyn(0x) - Rsyn(0y) = (5a + 3b + 5) + 2(j - i - 3) - (3a + 5b - 5) = 2(a - b) + 2(j - i) + 4 \equiv 0 \pmod{4n}.$$ 

From the representation of $x$ we have $a + 2 \leq b \leq a + j - i$. Hence, $4 \leq Rsyn(0x) - Rsyn(0y) \leq 2(j - i)$. Moreover, $2 \leq j - i \leq n - 5$, thus we have $4 \leq Rsyn(0x) - Rsyn(0y) \leq 2(n - 5)$, which is again a contradiction to the constraint that $Rsyn(0x) - Rsyn(0y) \equiv 0 \pmod{4n}$.

• Case 5: $\Delta_r(x) = \Delta_r(y) = 0$. This case is further divided depending on whether each error is $0001 \rightarrow 011$ or $1001 \rightarrow 110$. We only present one subcase as an example and the others can be proved analogously. Consider the subcase when both the error patterns in $x$ and $y$ are $0001 \rightarrow 011$. We have

$$x = (\ldots, 0, 0, 0, 0, 1, x_{[i+4,j]}, 0, 1, \ldots),$$

$$y = (\ldots, 0, 1, 1, y_{[i+2,j-2]}, 0, 0, 0, 0, 1, \ldots).$$

In this subcase (also other subcases), for every $i + 2 \leq k \leq j - 2$, the run index of $x_{k+2}$ in $R(0x)$ equals the run index of $y_k$ in $R(0y)$. Then we have

$$Rsyn(0x) - Rsyn(0y) = (5a + 3b + 3) - (3a + 5b + 3) = 2(a - b) \equiv 0 \pmod{4n}.$$ 

In this subcase, $a + 2 \leq b \leq a + j - i$. Therefore, $-2(n - 6) \leq -2(j - i - 1) \leq Rsyn(0x) - Rsyn(0y) \leq -4$. Moreover, $2 \leq j - i \leq n - 5$, $-2n + 12 \leq Rsyn(0x) - Rsyn(0y) \leq -4$, which is again a contradiction to the constraint that $Rsyn(0x) - Rsyn(0y) \equiv 0 \pmod{4n}$.

To sum up, we have assumed that we have two distinct codewords $x, y \in C_{3,1}(n; a, b, c, d)$ and $z \in F_{2}^{n-2}$ can be derived from both $x$ and $y$ by a $000 \rightarrow 1$ error. However, in all cases we can deduce that $Rsyn(0x) - Rsyn(0y)$ is nonzero modulo $4n$, which contradicts to the definition of the code. Thus, we have proven that $C_{3,1}(n; a, b, c, d)$ is an error-correcting code against a $000 \rightarrow 1$ error.

**Lemma 7:** $C_{3,1}(n; a, b, c, d)$ is an error-correcting code against $010 \rightarrow 1$ errors.

**Proof:** The proof follows the same way as the previous lemma. Suppose that we have two distinct codewords $x, y \in C_{3,1}(n; a, b, c, d)$ and $z \in F_{2}^{n-2}$ can be derived from both $x$ and $y$ by a $010 \rightarrow 1$ error. Then $x$ and $y$ should be of the form

$$x = (x_{[i-1,j]}, 0, 1, 0, x_{[i+3,j+1]}, 1, x_{[j+3,n]}),$$

$$y = (y_{[i-1,j]}, 1, y_{[i+1,j-1]}, 0, 1, 0, y_{[j+3,n]}),$$

where $x_k = y_k$ when $1 \leq k \leq i - 1$ and $j + 3 \leq k \leq n$, $x_{k+2} = y_k$ when $i + 1 \leq k \leq j - 1$.

First we need to observe the two values $\Delta_r(x), \Delta_r(y)$.

• If $i = 1$, i.e., the $(3,1)$-burst starts at the very beginning, then depending on whether $x_4 = 0$ or $x_4 = 1$, the error pattern is either $0100 \rightarrow 10$ or $0101 \rightarrow 11$ and thus $\Delta_r(x)$ is 1 or 3.

• If $i = n - 2$, i.e., the $(3,1)$-burst starts at the end, then depending on whether $x_{n-3} = 0$ or $x_{n-3} = 1$, the error pattern is either $0010 \rightarrow 01$ or $1010 \rightarrow 11$ and thus $\Delta_r(x)$ is 1 or 3.

• Otherwise, consider the four different cases for $x_{i-1}$ and $x_{i+3}$. The error pattern is $0010 \rightarrow 0101 \rightarrow 11101011 \rightarrow 1111$ and the corresponding value of $\Delta_r(x)$ is 0, 2, 2, and 4.

Based on the observations of $\Delta_r(x), \Delta_r(y)$ we then break into several cases and in each case we will derive a contradiction by analyzing $Rsyn(0x) - Rsyn(0y)$. Since the whole framework is similar as the previous lemma, we omit some details and only present the calculations for $Rsyn(0x) - Rsyn(0y)$.

• Case 1: $\Delta_r(x) = \Delta_r(y) = 1$.

$$x = (0, 1, 0, 0, x_{[5,n-2]}, 0, 1),$$

$$y = (1, 0, y_{[3,n-4]}, 0, 0, 1, 0).$$

Then $Rsyn(0x) - Rsyn(0y) = -2r(x) + 4 \neq 0$, since $r(x) \geq 4$.

• Case 2: $\Delta_r(x) = \Delta_r(y) = 3$.

$$x = (0, 1, 0, 1, x_{[5,n-2]}, 1, 1),$$

$$y = (1, 1, y_{[3,n-4]}, 1, 0, 1).$$

This case is a little bit special and deserves to be analyzed in detail. We can compute that $Rsyn(0x) - Rsyn(0y) = 2n - 2r(x) - 4$. Suppose it is zero modulo $4n$, then $n = r(x) + 2$. Since the first symbol of $x$ is 0 and the last symbol of $x$ is 1, then the number of runs $r(x)$ must be even and thereby $n$ must be even. As $\sum_{k=1}^{n/2} x_{2k-1} = 1 + k_{2k-1}, \sum_{k=1}^{n/2} y_{2k-1} = 3 + k_{2k-1}$ and $\sum_{k=1}^{n/2} x_{2k-1} = \sum_{k=1}^{n/2} y_{2k-1}$, we can get $\sum_{k=1}^{n/2} x_{2k-1} - \sum_{k=1}^{n/2} y_{2k-1} \equiv 0 \pmod{4}$ which is a contradiction to $\sum_{k=1}^{n/2} x_{2k-1} - \sum_{k=1}^{n/2} y_{2k-1} \equiv 0 \pmod{4}$. Therefore, $Rsyn(0x) - Rsyn(0y) = 2n - 2r(x) - 4 \neq 0 \pmod{4n}$.

• Case 3: $\Delta_r(x) = \Delta_r(y) = 0$ and the error pattern must be $0010 \rightarrow 010$. Hence,

$$x = (\ldots, 0, 0, 1, \ldots),$$

$$y = (\ldots, 0, 0, 1, \ldots).$$

Then $Rsyn(0x) - Rsyn(0y) = (2a - b) \neq 0 \pmod{4n}$.

• Case 4: $\Delta_r(x) = \Delta_r(y) = 4$ and the error pattern must be $1010 \rightarrow 111$. Hence,

$$x = (\ldots, 1, 0, 1, \ldots),$$

$$y = (\ldots, 1, 0, 1, \ldots).$$
Then $Rsyn(0x) - Rsyn(0y) = 2(a - b) + 4(j - i) + 8$.
Since $a + 4 \leq b \leq a + j - i + 2$, we have $2(j - i) + 4 \leq Rsyn(0x) - Rsyn(0y) \leq 4(j - i)$, so $Rsyn(0x) - Rsyn(0y) \neq 0$ (mod $4n$).

- Case 5: $\Delta(x) = \Delta(y) = 2$. This case is further divided depending on whether each error is 00101 or 01101.

Then $Rsyn(0x) - Rsyn(0y) = 2(a - b) + 2(j - i)$.
Suppose it is zero, then $b - a = j - i$. Since the symbols with run index $a$ are $0$s and symbols with run index $b$ are $1$s, $b - a$ must be odd and thus $j - i$ is odd. The number of symbols from $x_{i+j}$ to $x_j$ is $j - i - 3$ and thus is even. WLOG let $i$ be odd, then similarly as Case 2 we can analyze the sum of the odd coordinates and derive

$$\sum_{k=1}^{n/2} x_{2k-1} - \sum_{k=1}^{n/2} y_{2k-1} \equiv 1 - 3 \equiv 2 \pmod{4},$$

which is a contradiction to $\sum_{k=1}^{n/2} x_k - \sum_{k=1}^{n/2} y_k \equiv 0 \pmod{4}$. As a consequence, $Rsyn(0x) - Rsyn(0y) \neq 0$ (mod $4n$).

To sum up, we have assumed that there are two distinct codewords $x, y \in C_{3,1}(n; a, b, c, d)$ and $z \in \mathbb{F}_2^{n - 2}$ can be derived from both $x$ and $y$ by a 010 → 1 error. However, in all cases we can deduce that $Rsyn(0x) - Rsyn(0y)$ is nonzero modulo $4n$, which contradicts to the definition of the code. Thus, we have proven that $C_{3,1}(n; a, b, c, d)$ is an error-correcting code against a 010 → 1 error.

We close this section by summarizing our construction of the code $C_{3,1}(n; a, b, c, d)$:

- The decoding process starts with the observations of $\Delta_{odd}(u)$ and $\Delta_{even}(u)$.
- If we determine that the error can be seen as a 2-burst-deletion, then according to Levenshtein [16] our code is capable of correcting a 2-burst-deletion error.
- Otherwise, we can determine the error pattern, which is one out of $\{000 → 1, 010 → 1, 111 → 0, 101 → 0\}$.
- Whatever the error pattern is, our code can correct this type of error (two models are proved via Lemmas 6 and 7 and the other two follow a similar idea).

Finally, by the pigeonhole principle, we may find suitable parameters $a \in \mathbb{Z}_{4n}$, $b, c \in \mathbb{Z}_4$, and $d \in \mathbb{Z}_3$ and find a code with size at least $2^{n/2 - 80}$ and thus its redundancy is at most $\log(320n) < \log n + 9$. Note that we have proved that the lower bound of the redundancy of $(3, 1)$-burst correcting codes is $\log n + 2$, so our construction is optimal up to an additive constant. In sum, in this section we have proved the following.

**Theorem 9:** There exist choices of $a \in \mathbb{Z}_{4n}$, $b, c \in \mathbb{Z}_4$, and $d \in \mathbb{Z}_3$, such that the code $C_{3,1}(n; a, b, c, d)$ is a $(3, 1)$-burst correcting code with redundancy at most $\log n + 9$.

**VII. CONCLUSION AND FUTURE WORK**

In this paper we study $(t, s)$-burst correcting codes. First we prove the equivalence between $(t, s)$-burst correcting codes and $(s, t)$-burst correcting codes. Then we present a sphere-packing type upper bound of $(t, s)$-burst correcting codes, leading to a lower bound of its optimal redundancy. We present a construction of $(t, s)$-burst correcting codes for $t \geq 2s$, with redundancy $\log n + (t - s - 1) \log \log n + O(1)$. Comparing our general construction and the lower bound of redundancy, there is only a $\log \log n$ gap. We manage to close this gap for $t = 3, s = 1$ by giving a construction of $(3, 1)$-burst correcting codes with redundancy at most $\log n + 9$.

Here is a brief remark on our general construction for $t \geq 2s$. If we want to generalize the construction for the range $s \leq t < 2s$, then by viewing the codeword as an array of size $(t - s) \times \frac{n}{2-t}$, each row will suffer from more than a $(2, 1)$-burst. To be more specific, if $\frac{n}{2-t} < \frac{s}{2-t} < \frac{n}{2-t} + 1$, then in the array representation a row may suffer from a $(p, p-1)$-burst. Therefore, good constructions for codes against a $(p, p-1)$-burst for $p \geq 3$ will be useful tools in analyzing the general problem of $(t, s)$-burst correcting codes.

Another interesting problem is to analyze the redundancy of $(\leq t, \leq s)$-burst correcting codes and find explicit constructions of such codes. Following Theorem 5, a trivial solution is to use the intersection of a given family of $(t', s')$-burst correcting codes for $(t', s') \in \{(t, 0), (t, 1), \ldots, (t, s), (t - 1, s), \ldots, (s, s)\}$. Nontrivial constructions of $(\leq t, \leq s)$-burst correcting codes are considered for future research.

**ACKNOWLEDGMENT**

The authors would like to thank the editor and the reviewers for their constructive feedback and helpful suggestions which are very helpful to the improvement of the presentation of this article.

**REFERENCES**

[1] R. Bitar, S. K. Hanna, N. Polynskii, and I. Vorobyev, “Optimal codes correcting localized deletions,” 2021, arXiv:2105.02298.
[2] J. Brakensiek, V. Guruswami, and S. Zahror, “Efficient low-redundancy codes for correcting multiple deletions,” IEEE Trans. Inf. Theory, vol. 64, no. 5, pp. 3403–3410, May 2018.
[3] 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 Trans. Inf. Theory, vol. 67, no. 6, pp. 3438–3451, Jun. 2021.
[4] L. Cheng, T. G. Swart, H. C. Ferreira, and K. A. S. Abdel-Ghaffar, “Codes for correcting three or more adjacent deletions or insertions,” in Proc. IEEE Int. Symp. Inf. Theory, Jun. 2014, pp. 1246–1250.
[5] B. K. Dass, “On a burst-error correcting code,” J. Inf. Optim. Sci., vol. 1, no. 3, pp. 291–295, Jan. 1980.
[6] L. Dolecek and V. Anantharam, “Using Reed–Müller RM (1, m) codes over channels with synchronization and substitution errors,” IEEE Trans. Inf. Theory, vol. 53, no. 4, pp. 1430–1443, Apr. 2007.
[7] Y. Dong, F. Sun, Z. Ping, Q. Ouyang, and L. Qian, “DNA storage: Research landscape and future prospects,” Nat. Sci. Rev., vol. 7, no. 6, pp. 1092–1107, Jun. 2020.
[8] R. Gabrys, V. Guruswami, J. Ribeiro, and K. Wu, “Beyond single-deletion correcting codes: Substitutions and transpositions,” IEEE Trans. Inf. Theory, vol. 69, no. 1, pp. 169–186, Jan. 2023.
[9] R. Gabrys and F. Sala, “Codes correcting two deletions,” IEEE Trans. Inf. Theory, vol. 65, no. 2, pp. 965–974, Feb. 2019.
[10] R. Gabrys, E. Yaakobi, and O. Milenkovic, “Codes in the Damerau distance for deletion and adjacent transposition correction,” *IEEE Trans. Inf. Theory*, vol. 64, no. 4, pp. 2550–2570, Apr. 2018.

[11] V. Guruswami and J. Håstad, “Explicit two-deletion codes with redundancy matching the existential bound,” *IEEE Trans. Inf. Theory*, vol. 67, no. 10, pp. 6384–6394, Oct. 2021.

[12] S. Kas Hanna and S. El Rouayheb, “Guess & check codes for deletions, insertions, and synchronization,” *IEEE Trans. Inf. Theory*, vol. 65, no. 1, pp. 3–15, Jan. 2019.

[13] K. A. S. Immink, *Coding Techniques for Digital Recorders*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1991.

[14] A. Lenz and N. Polyanskiy, “Optimal codes correcting a burst of deletions of variable length,” in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2020, pp. 757–762.

[15] V. I. Levenshtein, “Binary codes capable of correcting deletions, insertions and reversals,” (in Russian), *Doklady Akademii Nauk SSSR*, vol. 163, no. 4, pp. 845–848, 1965.

[16] Ziyang Lu is currently pursuing the Ph.D. degree with Shandong University, Qingdao, Shandong, China. His research interests include combinatorics and coding theory and their interactions.

[17] Yiwei Zhang (Member, IEEE) received the B.A. and Ph.D. degrees in mathematics from Zhejiang University, Hangzhou, Zhejiang, China, in 2011 and 2016, respectively. From 2016 to 2017, he was a Post-Doctoral Researcher with the School of Mathematical Sciences, Capital Normal University, Beijing, China. From 2017 to 2019, he was a Post-Doctoral Researcher with the Department of Computer Science, Technion—Israel Institute of Technology, Haifa, Israel. He is currently a Professor with the School of Cyber Science and Technology, Shandong University, Qingdao, Shandong, China, where he is also with the Key Laboratory of Cryptologic Technology and Information Security, Ministry of Education. His current research interests include coding theory and information security.