Article

A Quaternary Code Correcting a Burst of at Most Two Deletion or Insertion Errors in DNA Storage

Thi-Huong Khuat [†] and Sunghwan Kim ^{*,†}

Department of Electrical, Electronic and Computer Engineering, University of Ulsan, Ulsan 44610, Korea; khuathuong090887@gmail.com

* Correspondence: sungkim@ulsan.ac.kr

† These authors contributed equally to this work.

Abstract: Due to the properties of DNA data storage, the errors that occur in DNA strands make error correction an important and challenging task. In this paper, a new code design of quaternary code suitable for DNA storage is proposed to correct at most two consecutive deletion or insertion errors. The decoding algorithms of the proposed codes are also presented when one and two deletion or insertion errors occur, and it is proved that the proposed code can correct at most two consecutive errors. Moreover, the lower and upper bounds on the cardinality of the proposed quaternary codes are also evaluated, then the redundancy of the proposed code is provided as roughly $2 \log_4 8n$.

Keywords: DNA storage; quaternary code; deletion error; insertion error; consecutive errors

Citation: Khuat, T.-H.; Kim, S. A Quaternary Code Correcting a Burst of at Most Two Deletion or Insertion Errors in DNA Storage. *Entropy* **2021**, *23*, 1592. <https://doi.org/10.3390/e23121592>

Academic Editor: Matteo Convertino

Received: 18 October 2021

Accepted: 26 November 2021

Published: 27 November 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

In recent years, because of its huge capacity and excellent durability, deoxyribonucleic acid (DNA) storage is becoming attractive for future long-term data storage [1–3]. However, during the processes of DNA storage, the molecule can be faced with errors that do not normally occur in traditional storage devices such as deletion and insertion errors [4]. Therefore, research to address deletion and insertion errors is extremely significant in DNA storage, and error-correcting codes for the errors have been studied. Our work focuses on the codes capable of correcting multiple deletion or insertion errors in DNA storage.

For correcting one deletion or insertion error in binary codes, Varshamov–Tenengolts (VT) codes were first proposed in [5] and in the same year the modified VT code construction was provided in [6] to correct a single deletion, insertion or substitution error. Shortly thereafter, to deal with more than a single error, Levenshtein extended the VT code to a binary code that can correct at most two consecutive deletion or insertion errors [7]. In [8], a binary codeword was arranged as an array with b rows and each row was a binary VT codeword so that this construction could correct a burst of the size of exactly b deletion or insertion errors (with any fixed $b \geq 2$). Then, the authors of [9] proposed a binary shifted-Varshamov–Tenengolts (SVT) code to obtain an improved construction which still corrects exactly b errors but with a lower redundancy than one in [8]. From the obviously efficient correction and low redundancy of the VT codes, the authors in [10,11] proposed a method of the linear-time encoders to implement the binary VT code which satisfies the homopolymer run and Guanine-Cytosine (GC)-content constraints [12,13] among important properties of a DNA strand. However, the binary VT codes used in these linear-time encoders correct a single nucleotide of a DNA strand. With a similar approach as [10,11], but to correct a burst of size exactly b deletions or insertions of DNA symbols, the authors of [14] applied the encoder of the binary modified VT code in [6] and binary SVT codes in [9]. Then, by interleaving bits of binary VT codewords and binary SVT codewords, the work [9] obtained a binary code construction that can correct a burst error of size exactly $2b$, and finally, the codeword of this construction was translated to DNA symbols.

A non-binary VT code was first proposed in [15], and a non-binary SVT code was proposed in [16]. The codes were defined over a q -ary alphabet for any $q > 2$. With the similar property of the binary codes, the q -ary VT and q -ary SVT codes can correct a single deletion or insertion symbol. To correct multiple errors, the construction in [8,9] can be applied to obtain a q -ary code that can correct a burst of size exactly b of deletion or insertion errors. However, designing q -ary VT codes that can correct multiple deletion or insertion errors has been an interesting problem [17]. Recently, there were some works [18–21] focused on code design to correct exact multiple errors but the efficient design for q -ary codes (or even quaternary codes) that can correct a burst of at most b deletion or insertion errors is still an open problem. The authors of [22] proposed a non-binary code correcting at most two consecutive deletions with redundancy $\log n + \log q (\log (\log n + 6) + \log 6) + 3$. In [22], the authors used the construction method in [9] with one binary code in [7] and a modified of it in interval P . However, we propose a quaternary code which is suitable for robust DNA storage and can correct at most two consecutive deletion or insertion errors with the direct construction. Moreover, the redundancy of the proposed code is improved than [22].

As to the cardinality of VT codes, for about 50 years, a lower bound of size of the best class of VT codes can be achieved, but an upper bound is rarely provided even in binary case. The author in [23] used Mixed Integer Linear Programming (MILP) relaxation technique to obtain the tighter upper bound of the binary VT code, for example, with the length $n = 11$, the maximum size of one deletion code was calculated as 173. Moreover, the conjecture about maximum size of VT code for all n was also provided. However, in this work, we focus on the correction error capability of the proposed code design, then we use the previous methods in [7,15] to evaluate lower bound and upper bound of the proposed code design.

In our work, we have extended binary codes based on the results of [7], by adding two constraints to determine the exact values and positions of the errors in the quaternary sequence. By mathematically analyzing the possible cases of errors, we propose decoding algorithms to prove the error correction capability of this code design. We note that the main concern in this work is the error correction capability of quaternary code design, not focus on constraints in DNA storage. It is assumed that the combination design of error correction code and constraints of DNA storage was already done by other algorithms [11,24]. The main contributions in this paper can be summarized as follows.

- We propose a quaternary code design that is suitable for the deletion or insertion channel, especially for mapping $0 \leftrightarrow A$, $1 \leftrightarrow C$, $2 \leftrightarrow T$, and $3 \leftrightarrow G$. This proposed design is directly applicable to sequencing in DNA storage. Furthermore, this proposed code can correct at most two consecutive deletion or insertion errors.
- We propose two decoding algorithms for this proposed code to correct one deletion and two consecutive deletion errors. For the decoding of insertion errors, some differences between the deletion and insertion cases are shown and the important functions for correcting the insertion error are also presented in Appendix.
- We provide the lower bound and evaluate upper bound of the proposed code design. The redundancy of the proposed code design is also calculated to be at most $2 \log_4 8n$.

This paper is organized as follows. In Section 2, we list basic notations and definitions used in the rest of the paper and we briefly present previous binary and quaternary code constructions to correct one and two consecutive deletions. Then, Section 3 contains the proposed code construction, a proof of the correction capability, and the bounds of the cardinality for the proposed quaternary code. Section 4 provides a discussion and, finally, conclusion is presented in Section 5 of this paper.

2. Preliminaries and Previous Works

2.1. Notation and Definition

Let $\mathbb{F}_2^n$ and $\mathbb{F}_4^n$ be the set of binary and quaternary sequences of length n , respectively. Let a quaternary codeword with length n be defined as $c = (c_1, c_2, \dots, c_n) \in \mathbb{F}_4^n$. Then, a

modified sequence c_l^{n-b} of the sequence c is defined as $c_l^{n-b} = (c_1, c_2, \dots, c_{l-2}, c_{l-1}, c_{l+b}, c_{l+b+1}, \dots, c_n) \in \mathbb{F}_4^{n-b}$, where $c_l, c_{l+1}, \dots, c_{l+b-1}$ are deleted in c . Similarly, a sequence c_l^{n+b} of the sequence c is also defined as $c_l^{n+b} = (c_1, c_2, \dots, c_{l-1}, h_1, h_2, \dots, h_b, c_l, c_{l+1}, \dots, c_n) \in \mathbb{F}_4^{n+b}$, where $h_1, h_2, \dots, h_b$ are inserted from l -th position in c .

For a binary sequence $x = (x_1, x_2, \dots, x_n) \in \mathbb{F}_2^n$, we can consider a sequence $0x$ with length $n + 1$, where $0x = (0, x_1, x_2, \dots, x_n) \in \mathbb{F}_2^{n+1}$. For simplicity, the sequence $0x$ with length $n + 1$ is regarded as having a starting value of $x_0 = 0$. For example, a binary sequence x with length 10 is given as $x = (0, 0, 1, 0, 0, 1, 1, 1, 1, 1)$. For convenience, the binary sequence notation can be changed to $x = 0010011111$. In the rest of this paper, these two notations are used as the same meaning, so there is a binary sequence $0x$, with length 11, as $0x = 00010011111$. Then, the run-length vector r denotes the number of zeros and ones run-length in $0x$. In addition, the binary sequence $0x$ is composed of four runs $u_0 u_1 u_2 u_3$, which are $u_0 = 000$, $u_1 = 1$, $u_2 = 00$, and $u_3 = 11111$. Herein, for a non-negative integer k , the zeros and ones runs are denoted u_{2k} and u_{2k+1} , respectively. Then, the run-length vector r of $0x$ is $r = (r_0, r_1, r_2, r_3) = (3, 1, 2, 5)$.

Let $\|r\|$ be the total number of elements in the run-length vector r of $0x$, corresponding to the total number of runs in $0x$. Then, from the run-length vector, the run-syndrome of the binary sequence $0x$ is defined as

$$Rsyn(0x) = \sum_{i=0}^{\|r\|-1} ir_i. \quad (1)$$

In the previous example, for $0x = 00010011111$, since the run-length vector r is $(3, 1, 2, 5)$, $Rsyn(0x) = \sum_{i=0}^4 ir_i = 20$.

If the j -th bit of $0x$ belongs to the m -th run u_m , we define $k_{0x}(j)$ as the index of the run and $k_{0x}(j) = m$, for $1 \leq j \leq n$. Since the total number of elements of the run-length vector cannot exceed the length of $0x$, $\|r\|$ is bounded as

$$\|r\| \leq n + 1, \quad (2)$$

where the equality is satisfied if the binary sequence $0x = 0101010 \dots$.

From the previous example, the binary sequence $0 = 00010011111$, with length 11, has the run-length vector $r = (r_0, r_1, r_2, r_3) = (3, 1, 2, 5)$ and the total number of elements of the run-length vector of $0x$ is $\|r\| = 4 < n + 1 = 11$. For $1 \leq j \leq 10$, since the third bit in $0x$ belongs to the run $u_1 = 1$, the index of the run which the third bit belongs to is $k_{0x}(3) = 1$.

2.2. Previous Works

With the binary case, to the best of our knowledge, the VT code in [5] is the best code to correct a single deletion or insertion error and the modified VT code in [6] is the best code to correct a single deletion, insertion, or substitution error. To correct more than a single error, we briefly recap the binary code correcting at most two consecutive deletions from [7]. Moreover, we briefly present the deletion correction capability of the binary code in [7] when single deletion or two consecutive deletions occur.

Definition 1. For $0 \leq d \leq 2n - 1$, the binary code $C(n, 2)$ in [7] with length n is given as

$$C(n, 2) = \{g \in \mathbb{F}_2^n : Rsyn(0g) \equiv d \pmod{2n}\}. \quad (3)$$

The correction capability of the code in Definition 1 was also proved in [7]. From the length of the received sequences y , we can know that one or two consecutive bits are removed from the codeword g . If one deletion at the j -th bit or two consecutive deletions at the j -th and $(j + 1)$ -th bits occur, then y can be g_j^{n-1} or g_j^{n-2} .

To determine the position of the deleted bit, we first calculate the difference of the run-syndrome as $\Delta = d - Rsyn(0y) \pmod{2n}$. If one deletion error occurs, $\Delta = d -$

$Rsyn(0g_j^{n-1}) \bmod 2n$ and if two consecutive deletions occur, $\Delta = d - Rsyn(0g_j^{n-2}) \bmod 2n$. These values are used to identify the value and position j of the deleted bit if there is one deletion in the codeword g or the values and positions j and $j + 1$ of the two deleted bits in the case that two consecutive deletions occur in the codeword g .

However, in the quaternary case, there exists the code to correct a single deletion or insertion error. The overview of q -ary insertion and deletion-correcting codes with length n is briefly presented in Definitions 2 and 3. The VT code family, known as the set of the most basic codes for correcting a single deletion or insertion, is defined as follows [15]:

Definition 2. For $0 \leq a < n$ and $0 \leq e < q$, the q -ary VT code with length n , $VT_{a,e}(n, q)$ is defined as

$$VT_{a,e}(n, q) \triangleq \left\{ c \in \mathbb{F}_q^n : \sum_{i=0}^n (i-1)\alpha_i \equiv a \bmod n \right. \quad (4)$$

$$\left. \sum_{i=0}^n c_i \equiv e \bmod q \right\}, \quad (5)$$

$$\text{where } \alpha_1 = 1 \text{ and } \alpha_i = \begin{cases} 1, & \text{if } c_i \geq c_{i-1} \\ 0, & \text{if } c_i < c_{i-1} \end{cases} \text{ for } 1 < i \leq n.$$

From Definition 2, since the binary sequence $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_n)$ is strongly related to the q -ary sequence c , a deletion of the j -th symbol in the codeword c also leads to a deletion of the j -th bit in the binary sequence α . Hence, from the help of the binary sequence α , the q -ary sequence is finally corrected.

Similarly, the authors of [16] proposed a single deletion-correcting code that defined the q -ary SVT code.

Definition 3. For $0 \leq a \leq P$, $0 \leq e < q$, and $f \in \{0, 1\}$, the q -ary SVT code, $SVT_{a,e,f}(n, P, q)$, with length n is defined as

$$SVT_{a,e,f}(n, P, q) \triangleq \left\{ c \in \mathbb{F}_q^n : \sum_{i=1}^n i\alpha_i \equiv a \bmod (P+1) \right. \quad (6)$$

$$\left. \sum_{i=1}^n c_i \equiv e \bmod q \right\} \quad (7)$$

$$\left. \sum_{i=1}^n \alpha_i \equiv f \bmod 2 \right\}, \quad (8)$$

$$\text{where } \alpha_1 = 1 \text{ and } \alpha_i = \begin{cases} 1, & \text{if } c_i \geq c_{i-1} \\ 0, & \text{if } c_i < c_{i-1} \end{cases} \text{ for } 1 < i \leq n.$$

Compared to the construction of the q -ary VT code, since $\bmod (P+1)$ is used in the constraint (6) in Definition 3 instead of $\bmod n$, the redundancy of the q -ary SVT code is reduced from $\lceil \log_q(n+1) \rceil$ to $\lceil \log_q(2P+2) \rceil + 1$. The constraint (8) is added to imply that the binary sequence α belongs to the binary SVT code. Hence, similar to the correcting method in Definition 2, the q -ary SVT code in Definition 3 can correct one deletion in any position.

However, the q -ary VT code in Definition 2 and the q -ary SVT code in Definition 3 correct only a single deletion or insertion error, but cannot correct consecutive deletions or insertions in the sequence. To solve this drawback, we can convert the idea in [8,9] about a construction for the binary codes correcting a burst of deletion or insertion errors with a size of exactly b for $b \geq 2$, into the q -ary case. The q -ary codeword c with length

n is treated as a codeword array $A_b(c)$ with size $b \times \frac{n}{b}$ and the codeword is arranged column-by-column. Then, to reduce redundancy than in [8], the first row and each of the other $(b - 1)$ rows in the codeword array are encoded by a q -ary VT code and q -ary SVT code, respectively. From this construction, one deletion or insertion error in each row can be corrected by the q -ary VT code or q -ary SVT code, such that a burst of b consecutive deletion or insertion errors can be corrected.

For example, for correcting a burst of deletions of size two, the q -ary codeword c with length n is presented as a $2 \times \frac{n}{2}$ array $A_2(c)$, which is given by

$$A_2(c) = \begin{bmatrix} c_1 & c_3 & \cdots & c_{n-1} \\ c_2 & c_4 & \cdots & c_n \end{bmatrix}. \quad (9)$$

Since each row of $A_2(c)$ is protected by the q -ary VT code or SVT code with length $\frac{n}{2}$, the code from $A_2(c)$ can correct exactly two consecutive deletions.

To sum up the previous statements, to correct one or exactly two consecutive deletion or insertion errors, we can use the q -ary VT and SVT codes. However, quaternary code to correct at most two consecutive deletion or insertion errors has not been developed. In the following section, we propose a new code design of quaternary codes suitable for the DNA storage and these codes can correct at most two consecutive deletion or insertion errors.

3. Proposed Code Design

This section provides a new design for a quaternary code to correct at most two consecutive deletions or insertions symbols. The construction of the proposed code is given in Section 3.1. Sections 3.2 and 3.3 prove the correction capabilities of the presented code if one deletion occurs or two consecutive deletion errors occur, respectively. The decoding of insertion errors is presented in Section 3.4. The evaluation of a lower bound and an upper bound on the cardinality of the proposed code is derived in Section 3.5.

3.1. Code Construction

Exploring a new design for the quaternary code to correct one or two consecutive symbols, we explain the proposed code design as the following definition. In the proposed code design, the binary sequence which has the same length and related to the quaternary sequence is used to construct the constraints for the proposed code.

Definition 4. For $0 \leq a \leq n$, $0 \leq d \leq 2n - 1$, and $0 \leq e < 4$, a quaternary code $C(n, 4)$ has a codeword $c = (c_1, c_2, \dots, c_n)$. First, we can consider a mapping from the quaternary codeword c to a binary sequence $x = (x_1, x_2, \dots, x_n)$ for $1 \leq i \leq n$ as,

$$x_i = \begin{cases} 0, & \text{if } c_i = 0 \text{ or } c_i = 1 \\ 1, & \text{if } c_i = 2 \text{ or } c_i = 3 \end{cases}. \quad (10)$$

Then, the quaternary code $C(n, 4)$ which satisfies the following three conditions can correct at most two consecutive deletion or insertion errors.

$$C(n, 4) = \left\{ c \in \mathbb{F}_4^n : R_{\text{syn}}(0x) \equiv d \pmod{2n} \right. \quad (11)$$

$$\left. \sum_{i=1}^n ic_i \equiv a \pmod{(8n+1)} \right. \quad (12)$$

$$\left. \sum_{i=1}^n c_i \equiv e \pmod{4} \right\}. \quad (13)$$

The basic idea of the mapping (10) is that the quaternary codeword c corresponds to the binary sequence x with the same length n . Therefore, a deletion in the j -th position of the codeword c also leads to a deletion in the j -th position of the binary sequence x . For

example, if the received sequence is $\mathbf{y} = \mathbf{c}_j^{n-1} \in \mathbb{F}_4^{n-1}$, after using the mapping (10), we can obtain the binary sequence $\mathbf{x}_j^{n-1} \in \mathbb{F}_2^{n-1}$, which has one deletion error in the j -th position.

In Definition 4, the condition (11) is the same as the condition (3) in Definition 1 for $C(n, 2)$, which means that the sequence $\mathbf{x}$ is protected by a binary codeword of $C(n, 2)$. Therefore, decoding of the binary sequence $\mathbf{x}$ can be used for finding the positions of the deleted symbols and guessing the values of deleted symbols of codeword $\mathbf{c}$.

The two constraints (12) and (13) in Definition 4, which are not in Definition 1, are used to obtain the correcting property in the quaternary regime. Since from constraint (11), the possible positions of the deletion errors can be obtained; however, in the case there is more than one value which satisfies the constraint (11), the constraints (12), (13) are used to remove invalid values of the possible positions. The constraint (13) is added to determine exactly the value of the deleted symbol and sum value of two consecutive deleted symbols. Then, finally the position and the value of symbols satisfy 3 constraints (11), (12) and (13) will be unique and the resulting quaternary sequence will be corrected. For example, $n = 10, d = 0, a = 0$ and $e = 0$, the binary sequence is corrected as $\mathbf{x} = \underline{11}0000111$, the underlined bits are the bits which are inserted to correct $\mathbf{x}$. From the mapping (10), the possible quaternary sequence can be $\mathbf{c} = 0300011322$, $\mathbf{c} = 0200011322$, $\mathbf{c} = 0310011322$, or $\mathbf{c} = 0210011322$. If there are no constraints (12), (13), the decoder cannot output the corrected quaternary sequence. Therefore, the constraints (12), (13) exclude the invalid quaternary sequences as described in Table 1, then the output is the unique sequence $\mathbf{c} = 0300011322$.

Table 1. Correction capability of constraints (12), (13) when two consecutive deletions occur.

Possible Corrected Quaternary Sequences	Compared to a in Constraint (12)	Compared to e in Constraint (13)
0300011322	O	O
0200011322	X	X
0310011322	X	X
0210011322	X	O

3.2. Decoding Procedure for One Deletion Error

It is assumed that a transmitter and receiver share the parameters n, d, a, e of the code $C(n, 4)$ in Definition 4. Then, we first consider a case that one deletion error occurs in the codeword $\mathbf{c}$. For $1 \leq i \leq n$, if the j -th symbol in $\mathbf{c}$ is removed, we obtain a received sequence $\mathbf{y} = \mathbf{c}_j^{n-1} \in \mathbb{F}_4^{n-1}$, with length $n - 1$.

If the symbol at the j -th position is deleted, the constraint (13) can be rewritten as $\sum_{i=1}^{j-1} c_i + c_j + \sum_{i=j+1}^n c_i \equiv e \pmod{4}$. From the received sequence $\mathbf{y} = \mathbf{c}_j^{n-1} \in \mathbb{F}_4^{n-1}$, the constraint is given as $\sum_{i=1}^{n-1} y_i = \sum_{i=1}^{j-1} y_i + \sum_{i=j}^{n-1} y_i = \sum_{i=1}^{j-1} c_i + \sum_{i=j+1}^n c_i$. Thus, the value of the deleted symbol value c_j is calculated as $c_j = e - \sum_{i=1, i \neq j}^n c_i \pmod{4} = e - \sum_{i=j}^{n-1} y_i \pmod{4}$.

Next, we need to find the deletion position j . From the mapping (10) for the received sequence $\mathbf{y}$ to acquire the binary sequence $\mathbf{x}_j^{n-1}$ with length $n - 1$, $0\mathbf{x}_j^{n-1}$ is obtained as $0\mathbf{x}_j^{n-1} = (0, x_1, x_2, \dots, x_{j-2}, x_{j-1}, x_{j+1}, x_{j+2}, \dots, x_n)$. Then, the run-length vector $\mathbf{r}'$ is determined from $0\mathbf{x}_j^{n-1}$ and $Rsyn(0\mathbf{x}_j^{n-1}) = \sum_{i=0}^{\|\mathbf{r}'\|-1} i\mathbf{r}'_i \pmod{2n}$ in the constraint (11). As mentioned in Definition 1, when one deletion error occurs, the run-syndromes decreases by $\Delta = d - Rsyn(0\mathbf{x}_j^{n-1}) \pmod{2n}$.

To provide a proof for the correction capabilities of the proposed quaternary code in Definition 4, we develop Algorithm 1 as a correcting method in the case of one deletion symbol.

Algorithm 1 Correct one deletion symbol.

Input: $n, d, a, e, \mathbf{y} = \mathbf{c}_j^{n-1} \in \mathbb{F}_4^{n-1}$.
Output: $\mathbf{c} = (c_1, c_2, \dots, c_n) \in C(n, 4)$.
1: $c_j = e - \sum_{i=j}^{n-1} y_i \bmod 4$.
2: Get the binary sequence $0\mathbf{x}_j^{n-1}$ and the run-length vector $\mathbf{r}'$ of $0\mathbf{x}_j^{n-1}$.
3: Get the total number of elements of $\mathbf{r}'$ as $\|\mathbf{r}'\|$.
4: $\Delta = d - Rsyn(0\mathbf{x}_j^{n-1}) \bmod 2n$.
5: Set $j = 1$.
6: **while** $j \leq n$ **do**
7: **if** $\Delta < \|\mathbf{r}'\|$ **then**
8: **if** $k_{0\mathbf{x}_j^{n-1}}(j-1) = \Delta$ **then**
9: $\mathbf{c} = \mathbf{del_correct1}(n, a, \mathbf{y}, j, c_j)$
10: **else**
11: $j \leftarrow j + 1$
12: **end if**
13: **else**
14: **if** $k_{0\mathbf{x}_j^{n-1}}(j-1) + 2(n-j) - 1 = \Delta$ **then**
15: $\mathbf{c} = \mathbf{del_correct1}(n, a, \mathbf{y}, j, c_j)$
16: **else**
17: $j \leftarrow j + 1$
18: **end if**
19: **end if**
20: **end while**

Function 1 provides function **del_correct1** for Algorithm 1 to determine the deletion position, and then the output of Function 1 is the corrected quaternary sequence. In addition, in Function 1, Syn_new stands for the syndrome of the quaternary sequence after inserting the lost symbol c_j in the j -th position of $\mathbf{c}_j^{n-1}$.

Function 1: $\mathbf{c} = \mathbf{del_correct1}(n, a, \mathbf{y}, j, c_j)$

Input: $n, a, \mathbf{y}, j, c_j$.
Output: $\mathbf{c} = (c_1, c_2, \dots, c_n) \in C(n, 4)$.
1: $Syn_new = \sum_{i=1}^{j-1} ic_{j,i}^{n-1} + jc_j + \sum_{i=j+1}^{n-1} ic_{j,i}^{n-1} \bmod (8n+1)$
2: **if** $Syn_new = a$ **then**
3: $\mathbf{c} = (c_1, c_2, \dots, c_{j-1}, c_j, c_{j+1}, \dots, c_n)$
4: **else**
5: $j \leftarrow j + 1$
6: **end if**

Example 1: Let n, d, a , and e be 10, 0, 0, and 0, respectively. Assume that one deletion occurs at the sixth position of the codeword $\mathbf{c} = (0, 3, 0, 0, 0, 1, 1, 3, 2, 2) \in \mathbb{F}_4^{10}$. The received sequence $\mathbf{y}$ is $\mathbf{y} = \mathbf{c}_j^9 = (0, 3, 0, 0, 0, 1, 3, 2, 2)$. As mentioned in Algorithm 1, the value of the lost symbol is $c_j = e - \sum_{i=1}^{n-1} y_i \bmod 4 = 1$. From the mapping (10), we obtain the binary sequence $\mathbf{x}_j^9 = 010000111$. Then, the run-length vector of $0\mathbf{x}_j^9$ is $\mathbf{r}' = (2, 1, 4, 3)$ so $\|\mathbf{r}'\| = 4$ and the run-syndrome of $0\mathbf{x}_j^9$ is $Rsyn(0\mathbf{x}_j^9) = \sum_{i=0}^{5-1} ir'_i \bmod 20 = 18$. The change of the run-syndrome is computed as $\Delta = 0 - Rsyn(0\mathbf{x}_j^9) \bmod 20 = 2$.

For $1 \leq j \leq n$, since $\Delta < \|\mathbf{r}'\|$, following Algorithm 1, when $j = 6$, then $\Delta = k_{0\mathbf{x}_j^9}(j-1) = 2$. If inserting the lost symbol with $c_j = 1$ in the sixth position of the received sequence as $(0, 3, 0, 0, 0, 1, 1, 3, 2, 2)$, the syndrome of this quaternary sequence $Syn_new = \sum_{i=1}^{6-1} ic_{6,i}^9 + 6.1 + \sum_{i=6+1}^{10} ic_{6,i}^9 \bmod 81 = 0$ (equals to a). Thus, the deletion error of the quaternary sequence is recovered correctly.

3.3. Decoding Procedure for Two Deletion Errors

Suppose that the received sequence $\mathbf{y} = \mathbf{c}_j^{n-1} \in \mathbb{F}_4^{n-2}$ with length $n - 2$, where two consecutive symbols in the j -th and $(j + 1)$ -th positions of codeword $\mathbf{c} \in C(n, 4)$ are deleted.

The constraint (13) in Definition 4 can be rewritten as $\sum_{i=1}^{j-1} c_i + c_j + c_{j+1} + \sum_{i=j+2}^n c_i \equiv e \pmod{4}$, and it is easy to obtain as $c_j + c_{j+1} = e - \sum_{i=1, i \neq j, i \neq j+1}^n c_i \pmod{4}$, corresponding to $c_j + c_{j+1} = e - \sum_{i=1}^{n-2} c_{j,i}^{n-2} \pmod{4}$. Since $\sum_{i=1}^{n-2} y_i = \sum_{i=1}^{n-2} c_{j,i}^{n-2}$, we can rewrite $c_j + c_{j+1}$ as $c_j + c_{j+1} = e - \sum_{i=1}^{n-2} y_i \pmod{4}$.

From the mapping (10) for the received sequence $\mathbf{y}$, the binary sequence with length $n - 1$ is obtained as $0\mathbf{x}_j^{n-2} = (0, x_1, x_2, \dots, x_{j-2}, x_{j-1}, x_{j+2}, x_{j+3}, \dots, x_n)$. Then, the run-length vector $\mathbf{r}''$ of $0\mathbf{x}_j^{n-2}$ also can determine the run-syndrome of $0\mathbf{x}_j^{n-2}$ as $Rsyn(0\mathbf{x}_j^{n-2}) = \sum_{i=0}^{\|\mathbf{r}''\|-1} ir_i'' \pmod{2n}$. Thus, similar to the approach mentioned in Definition 1, the difference of the run-syndrome is computed as $\Delta = d - Rsyn(0\mathbf{x}_j^{n-2}) \pmod{2n}$.

To recover two deletion errors, we first recover the binary sequence $\mathbf{x}$ with length n from the binary sequence $\mathbf{x}_j^{n-2}$. The authors of [7] suggested the eight possible instances when two consecutive bits are deleted, as summarized in Table 2. However, in this work, we consider more instances which are 16 in total, and the remaining eight instances are listed in Table 3. Please note that in Algorithms 2 and A2, a notation $x_j = \bar{x}_{j-1}$ is used to imply that the reverse value of the $(j - 1)$ -th position is assigned to the bit at the j -th position. Thus, two notations $x_j = \bar{x}_{j-1}$ and $x_j \neq \bar{x}_{j-1}$ have the same meaning, and this means that two neighbor $(j - 1)$ -th and j -th bits have different values. For example, if $x_{j-1} = 1$, then $x_j = \bar{x}_{j-1} = 0$ or $x_j \neq \bar{x}_{j-1} = 0$.

Table 2. The eight possible instances in [7] for two consecutive deletion errors.

Conditions of $(x_{j-1}, x_j, x_{j+1}, x_{j+2})$ in [7]	$(x_{j-1}, x_j, x_{j+1}, x_{j+2})$ in [7]
$x_j = x_{j-1}, x_{j+1} = x_{j-1}, x_{j+2} \neq x_{j-1}$ if $j + 2 \leq n$	(1,1,1,0) (0,0,0,1)
$x_j = x_{j-1}, x_{j+1} \neq x_{j-1}, x_{j+2} \neq x_{j-1}$ if $j + 2 \leq n$	(1,1,0,0) (0,0,1,1)
$x_j \neq x_{j-1}, x_{j+1} \neq x_{j-1}, x_{j+2} = x_{j-1}$ if $j + 2 \leq n$	(1,0,0,1) (0,1,1,0)
$x_j \neq x_{j-1}, x_{j+1} = x_{j-1}, x_{j+2} = x_{j-1}$ if $j + 2 \leq n$	(0,1,0,0) (1,0,1,1)

Table 3. The eight possible instances are added in this work for two consecutive deletion errors.

Conditions of $(x_{j-1}, x_j, x_{j+1}, x_{j+2})$	$(x_{j-1}, x_j, x_{j+1}, x_{j+2})$
$x_j = x_{j-1}, x_{j+1} = x_{j-1}, x_{j+2} = x_{j-1}$ if $j + 2 \leq n$	(1,1,1,1) (0,0,0,0)
$x_j = x_{j-1}, x_{j+1} \neq x_{j-1}, x_{j+2} = x_{j-1}$ if $j + 2 \leq n$	(1,1,0,1) (0,0,1,0)
$x_j \neq x_{j-1}, x_{j+1} \neq x_{j-1}, x_{j+2} \neq x_{j-1}$ if $j + 2 \leq n$	(1,0,0,0) (0,1,1,1)
$x_j \neq x_{j-1}, x_{j+1} = x_{j-1}, x_{j+2} \neq x_{j-1}$ if $j + 2 \leq n$	(0,1,0,1) (1,0,1,0)

In an analysis approach similar to [7], for $1 \leq j \leq n - 1$, there are four possible deleted bit pairs $(x_j, x_{j+1}) = (0, 0), (0, 1), (1, 0)$, and $(1, 1)$. Then, if $j + 2 \leq n$, we combine four possible cases of (x_j, x_{j+1}) with neighboring bits $(x_{j-1}, x_{j+2}) = (0, 0), (0, 1), (1, 0)$, and $(1, 1)$, and we need to consider 16 instances of $(x_{j-1}, x_j, x_{j+1}, x_{j+2})$.

From the above analysis, we develop Algorithm 2 for the proposed code to correct two consecutive deletion errors. In addition, in Algorithm 2, though it was not mentioned, the bit x_{j+2} is mathematically analyzed as an accompanied pair with x_{j-1} , as described above, to obtain the conditions, such as lines 9, 16, 27, 34 to determine the deleted positions.

Algorithm 2 Correct two consecutive deletion symbols

Input: $n, d, a, e, \mathbf{y} = \mathbf{c}_j^{n-2} \in \mathbb{F}_4^{n-2}$.
Output: $\mathbf{c} = (c_1, c_2, \dots, c_n) \in C(n, 4)$.
1: $c_j + c_{j+1} = e - \sum_{i=j}^{n-2} y_i \bmod 4$.
2: Get the binary sequence $0x_j^{n-2}$ and the run-length vector $\mathbf{r}''$ of $0x_j^{n-2}$.
3: Get the total number of elements of $\mathbf{r}''$ as $\|\mathbf{r}''\|$.
4: $\Delta = d - \text{Rsyn}(0x_j^{n-2}) \bmod 2n$.
5: Set $j = 1$.
6: **if** $\Delta \geq 2\|\mathbf{r}''\|$ **then**
7: **while** $j \leq n - 1$ **do**
8: **if** $\text{mod}(\Delta, 2) = 1$ **then**
9: **if** $2k_{0x_j^{n-2}}(j - 1) + 2(n - j) + 1 = \Delta$ **then**
10: $x_j = \bar{x}_{j-1}; x_{j+1} = x_{j-1}$
11: $\mathbf{c} = \text{del_correct2}(n, a, \mathbf{y}, j, x_j, x_{j+1}, c_j + c_{j+1})$
12: **else**
13: $j \leftarrow j + 1$
14: **end if**
15: **else**
16: **if** $2k_{0x_j^{n-2}}(j - 1) + 2(n - j) = \Delta$ **then**
17: $x_j = \bar{x}_{j-1}; x_{j+1} = \bar{x}_{j-1}$
18: $\mathbf{c} = \text{del_correct2}(n, a, \mathbf{y}, j, x_j, x_{j+1}, c_j + c_{j+1})$
19: **else**
20: $j \leftarrow j + 1$
21: **end if**
22: **end if**
23: **end while**
24: **else**
25: **while** $j \leq n - 1$ **do**
26: **if** $\text{mod}(\Delta, 2) = 1$ **then**
27: **if** $2k_{0x_j^{n-2}}(j - 1) + 1 = \Delta$ **then**
28: $x_j = x_{j-1}; x_{j+1} = \bar{x}_{j-1}$
29: $\mathbf{c} = \text{del_correct2}(n, a, \mathbf{y}, j, x_j, x_{j+1}, c_j + c_{j+1})$
30: **else**
31: $j \leftarrow j + 1$
32: **end if**
33: **else**
34: **if** $2k_{0x_j^{n-2}}(j - 1) = \Delta$ **then**
35: $x_j = x_{j-1}; x_{j+1} = x_{j-1}$
36: $\mathbf{c} = \text{del_correct2}(n, a, \mathbf{y}, j, x_j, x_{j+1}, c_j + c_{j+1})$
37: **else**
38: $j \leftarrow j + 1$
39: **end if**
40: **end if**
41: **end while**
42: **end if**

To clarify the explanation of the function **del_correct2** for Algorithm 2 in Section 3.3, we provide the detail in Function 2. In Function 2, *Syn_new* implies the syndrome of the quaternary sequence after inserting the lost symbols c_j and c_{j+1} in the j -th and $(j + 1)$ -th

position of c_j^{n-2} . If the value of Syn_new equals to the parameter of syndrome a , we infer that the quaternary sequence is retrieved successful.

Function 2: $c = \text{del_correct2}(n, a, y, j, c_j + c_{j+1})$

Input: $n, a, y, j, x_j, x_{j+1}, c_j + c_{j+1}$.

Output: $c = (c_1, c_2, \dots, c_n) \in C(n, 4)$.

```

1: Using mapping (10) to obtain  $c_j$ , then  $c_{j+1} = c_j + c_{j+1} - c_j$ .
2:  $Syn\_new = \sum_{i=1}^{j-1} i c_{j,i}^{n-2} + j c_j + (j+1) c_{j+1} + \sum_{i=j+2}^{n-2} i c_{j,i}^{n-2} \bmod (8n+1)$ 
2: if  $Syn\_new = a$  then
3:    $c = (c_1, c_2, \dots, c_{j-1}, c_j, c_{j+1}, \dots, c_n)$ 
4: else
5:    $j \leftarrow j+1$ 
6: end if

```

Example 4: Let n, d, a , and e be 10, 0, 0, and 0, respectively. It is assumed that two consecutive deletions occur at the seventh and eighth position of the codeword $c = (0, 3, 0, 0, 0, 1, 1, 3, 2, 2) \in \mathbb{F}_4^{10}$ and the received the quaternary sequence is $y = c_j^8 = (0, 3, 0, 0, 0, 1, 2, 2)$. As mentioned in Algorithm 2, the sum of the values of the two deleted symbols is $c_j + c_{j+1} = e - \sum_{i=1}^{n-2} y_i \bmod 4 = e - \sum_{i=1}^{n-2} c_{j,i}^8 \bmod 4 = 0$.

From the mapping (10) for c_j^8 , the binary sequence $0x_j^8$ and r'' are $0x_j^8 = 001000011$ and $r'' = (2, 1, 4, 2)$, respectively. Then, $\|r''\| = 4$ and the run-syndrome $Rsyn(0x_j^8) = 15$. The difference of run-syndrome is calculated as $\Delta = 0 - Rsyn(0x_j^8) \bmod 20 = 5$.

From Algorithm 2, since $\Delta < 2\|r''\|$ and $\Delta \bmod 2 = 1$, for $1 \leq j \leq n-1$, the value $j = 7$ satisfies the equation $\Delta = 2k_{0x_j^8}(j-1) + 1 = 2k_{0x_j^8}(6) + 1 = 5$. Thus, as mentioned in line 28 of Algorithm 2, we obtain $x_7 = 0$ and $x_8 = 1$, and the corrected binary sequence is 0100000111.

Applying mapping (10) to the binary sequence 0100000111 and $c_j + c_{j+1} = 0$, the two deleted symbols (c_7, c_8) are determined as $(1, 3)$. The syndrome Syn_new of the quaternary sequence when inserting $(c_7, c_8) = (1, 3)$ into c_j^8 is 0, which equals the syndrome a of codeword c . Thus, finally the recovered quaternary sequence is $(0, 3, 0, 0, 0, 1, 1, 3, 2, 2)$.

Algorithms 1 and 2 are proved using an exhaustive search strategy to show that the proposed code can correct at most two consecutive deletion symbols. However, as mentioned in [9], deletion-correcting codes are not always successful in identifying the exact location of the deleted symbols. For example, if an all-zero codeword is sent and one deletion error occurs, to find value of the deleted symbol is easy but it is impossible to find the exact position of the deleted symbol. Even though the exact position cannot be detected, the codeword can be successfully recovered by inserting a zero symbol in any position. This means that when the exact index of the deleted error is not detected but the run index which the deleted error belongs to is determined, the codeword can be successfully recovered by inserting one symbol in any position in the run.

If a codeword with a large run was sent and one deletion occurs in the large run, the proposed algorithm can always determine the value and the run index of the deleted symbol but rarely find the exact position of the deleted symbol in the run. In this case, we prioritize the proposed algorithm to output the first index in the detected run. Therefore, when a deletion error occurs in a large run and it is not possible to find the exact position in a codeword, the codeword of the proposed code will be successfully decoded by inserting the deleted symbol in the first index of the run.

3.4. Decoding Procedure for Insertion Errors

Since there is a similarity to the case of deletion errors, in this subsection, the correction capability of this proposed code for insertion errors is briefly presented. The received quaternary sequence y has a length that is one or two symbols larger than n , if one or

two consecutive insertion errors occur. Table 4 summarizes the different computations of decoding between insertion and deletion errors.

Table 4. The differences between insertion and deletion errors.

Content	One Insertion Error	Two Consecutive Insertion Errors	One Deletion Error	Two Consecutive Deletion Errors
Length of the received sequence	$n + 1$	$n + 2$	$n - 1$	$n - 2$
Sum of error symbol(s)	$\sum_{i=1}^{n+1} y_i - e \bmod 4$	$\sum_{i=1}^{n+2} y_i - e \bmod 4$	$e - \sum_{i=1}^{n-1} y_i \bmod 4$	$e - \sum_{i=1}^{n-2} y_i \bmod 4$
Difference of run-syndrome (mod 2n)	$Rsyn(0x_j^{n+1}) - d$	$Rsyn(0x_j^{n+2}) - d$	$d - Rsyn(0x_j^{n-1})$	$d - Rsyn(0x_j^{n-2})$

3.4.1. Correcting one Insertion Error

It is assumed that the received sequence with length $n + 1$ is $\mathbf{y} = \mathbf{c}_j^{n+1} = (c_1, c_2, \dots, c_{j-1}, h_1, c_j, c_{j+1}, \dots, c_n) \in \mathbb{F}_4^{n+1}$, this means that one symbol h_1 is inserted at the j -th position of the codeword $\mathbf{c} \in C(n, 4)$. The process to correct the received sequence $\mathbf{y}$ can be briefly presented by the following steps.

The first step is calculating the value of the inserted symbol h_1 in $\mathbf{y}$. The received sequence $\mathbf{y}$ has a sum of total symbols computed as $\sum_{i=1}^{n+1} y_i = \sum_{i=1}^{n+1} c_{j,i}^{n+1} = \sum_{i=1}^{j-1} c_i + h_1 + \sum_{i=j}^n c_i$, and then $\sum_{i=1}^{n+1} c_{j,i}^{n+1} \bmod 4 = \sum_{i=1}^n c_i + h_1 \bmod 4 = e + h_1 \bmod 4$. The value of the inserted symbol h_1 is calculated by $h_1 = \sum_{i=1}^{n+1} c_{j,i}^{n+1} - e \bmod 4$.

The second step is determining the insertion position j . From mapping (10), we obtain the binary sequence $0x_j^{n+1}$. From the binary sequence $0x_j^{n+1}$, we obtain the run-length vector of $0x_j^{n+1}$ and then calculate the difference of the run-syndrome by $\Delta = Rsyn(0x_j^{n+1}) - d \bmod 2n$. To determine the position j of the inserted symbol h_1 , in Appendix A, we provide Algorithm A1 and Function 3 for this step and the output is the corrected quaternary sequence.

3.4.2. Correcting Two Consecutive Insertion Errors

If two consecutive insertion errors occur at the j -th and $(j+1)$ -th positions of the codeword $\mathbf{c} \in C(n, 4)$, the received sequence is $\mathbf{y} = \mathbf{c}_j^{n+2} = (c_1, c_2, \dots, c_{j-1}, h_1, h_2, c_j, c_{j+1}, \dots, c_n) \in \mathbb{F}_4^{n+2}$ with length $n + 2$. From the received sequence $\mathbf{c}_j^{n+2}$ and a similar analysis as the one insertion case, the sum of the two inserted symbols is obtained as $h_1 + h_2 = \sum_{i=1}^{n+2} c_{j,i}^{n+2} - e \bmod 4 = \sum_{i=1}^{n+2} y_i - e \bmod 4$.

From the mapping (10) in Definition 4, since two consecutive symbols h_1, h_2 are inserted in $\mathbf{c}_j^{n+2}$ corresponding two consecutive bits are also inserted in the binary sequence $\mathbf{x}_j^{n+2}$, we can obtain the binary sequence $0x_j^{n+2}$. Thus, the run-syndrome of $0x_j^{n+2}$ is calculated by $\Delta = Rsyn(0x_j^{n+2}) - d \bmod 2n$. Algorithm A2 and Function 4 in Appendix B are provided to determine exact values of h_1, h_2 and the positions j and $j+1$ of the two consecutive insertion errors. Finally, h_1 and h_2 are removed from the sequence $\mathbf{c}_j^{n+2}$ to retrieve the codeword $\mathbf{c}$.

3.5. Cardinality of the Proposed Code

Since our main contribution is the correction code capability of the proposed code, then the lower bounds and upper bound of this code design is evaluated based on the previous methods in [7,15].

3.5.1. Lower Bound of the Code Cardinality

In [15] of Section IV, the lower bound of the code cardinality was determined by the potential values of the syndrome and checksum in the code construction. Hence, with the similar approach, by applying $d \in [0, 2n - 1]$, $a \in [0, 8n]$ and $e \in [0, 3]$, we can obtain the lower bound for the cardinality $m(n, 4)$ of the proposed code as

$$m(n, 4) \geq \frac{4^n}{8n(8n + 1)}. \quad (14)$$

The redundancy of the proposed code can be at most as below

$$n - \log_4 |C(n, 4)| \leq n - \log_4 \frac{4^n}{8n(8n + 1)} \approx 2 \log_4 8n. \quad (15)$$

3.5.2. Upper Bound of the Code Cardinality

Define $|M(n, 4)|$ as the cardinality of the quaternary code of length n , with a maximum possible number of codewords, which can correct at most two consecutive deletion or insertion errors. Similar to the method in [7], the upper bound of the cardinality of $|M(n, 4)|$ is evaluated as

$$|M(n, 4)| \leq |M_1(n, 4)| + |M_2(n, 4)|. \quad (16)$$

where $|M_1(n, 4)|$ is the number of codewords with length n such that the number of runs is larger than $(r + 1)$ (with r is an arbitrary number) and $|M_2(n, 4)|$ is number of codewords with length n such that the number of runs is not larger than $(r + 1)$. The equation (16) will be represented as

$$|M(n, 4)| \leq \frac{4^{n-2}}{2(r + 1) + 1} + 2 \cdot 4^{n - \lfloor \frac{n-2}{2} \rfloor} \sum_{j=0}^r \binom{\lfloor \frac{n-2}{2} \rfloor}{j}. \quad (17)$$

Let we set $r = \lfloor \frac{3 \lfloor \frac{n-2}{2} \rfloor - \sqrt{2 \lfloor \frac{n-2}{2} \rfloor \ln \lfloor \frac{n-2}{2} \rfloor}}{4} \rfloor$, and let n tends to infinity, then $2(r + 1) + 1 \approx \frac{3}{2}n$. Therefore, with $r \leq \frac{3 \lfloor \frac{n-2}{2} \rfloor}{4}$, the upper bound of the cardinality of the proposed code can be written as

$$|M(n, 4)| \leq \frac{2 \cdot 4^{n-2}}{3n}. \quad (18)$$

4. Discussion

In this section, we explain the results of our proposed code design and then discuss about the applications of the proposed code.

We provide a new design of quaternary codes to correct at most two consecutive deletion or insertion errors. From Algorithms 1 and 2 and Appendixes A and B the correction capabilities of this design with deletion and insertion errors are proved. Obviously, with this proposed code, we can consider $0 \leftrightarrow A$, $1 \leftrightarrow C$, $2 \leftrightarrow T$, and $3 \leftrightarrow G$ to directly construct or sequencing the DNA strands.

To deal with a burst of size of at most b (for any fixed $b \geq 3$) deletion or insertion errors, the intersection between the proposed code and the quaternary code can correct exactly $b - 2$ consecutive deletion or insertion errors.

For example, to correct a burst error of a size of at most $b = 3$, first, we create the code $C(n, 4)$ from Definition 4, which takes care of one or two consecutive deletion or insertion errors. Then, we create a q -ary VT code $VT_{a,e}(n, 4)$ from Definition 2 with $q = 4$ to correct a single error. Then, by intersecting $C(n, 4)$ and $VT_{a,e}(n, 4)$ we can obtain the expected quaternary code with length n , to deal with a burst error of size at most three. With given $b > 3$, we use the array code construction which is described in Section 2.2 to

create a quaternary code that can correct exactly $b - 2$ consecutive deletion or insertion errors. Through intersection of this code and our proposed code, a quaternary code that can correct at most $b > 3$ consecutive deletion or insertion errors can be obtained.

5. Conclusions

In this paper, we propose a new design of a quaternary code to correct at most two consecutive deletion or insertion errors with redundancy at most $2 \log_4 8n$ symbols. We also develop decoding algorithms for correcting one and two consecutive deletion or insertion errors in any quaternary sequences. Even though the results in this work provide significant applications for DNA storage and correction of multiple quaternary errors, there are still several open problems, such as code constructions which can correct at most b non-consecutive deletion or insertion errors and codes that can correct at most b deletion or insertion and substitution errors, for arbitrary b . Moreover, the optimal design when concatenation of constrained code and our proposed code for DNA-based data storage also needs to be considered.

Author Contributions: All authors discussed the contents of the manuscript and contributed to its presentation. T.-H.K. designed and implemented the proposed code construction and algorithms, wrote the paper under the supervision of S.K. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by Samsung Research Funding & Incubation Center of Samsung Electronics under Project Number SRFC-IT1802-09.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

DNA	Deoxyribonucleic acid
VT	Varshamov–Tenengolts
SVT	shifted-Varshamov–Tenengolts
GC-content	Guanine–Cytosine content
MILP	Mixed Integer Linear Programming

Appendix A. One Insertion Error Correction

As mentioned in Section 3.4, Algorithm A1 and the function `ins1_correct1` which is used in Algorithm A1 are given as follows.

Algorithm A1 Correct one insertion symbol

Input: $n, a, y, 0x_j^{n+1}$.
Output: $c = (c_1, c_2, \dots, c_n) \in C(n, 4)$.
1: Calculate Δ and h_1 as in Table 3.
2: Set $j = 1$.
3: **while** $j \leq n + 1$ **do**
4: **if** $\Delta < \|r'\|$ **then**
5: **if** $\text{mod}(\Delta, 2) = 1$ **then**
6: **if** $k_{0x_j^{n+1}}(j - 1) + 1 = \Delta$ **then**
7: $c = \text{ins1_correct1}(n, a, y, j, h_1)$
8: **else**
9: $j \leftarrow j + 1$
10: **end if**
11: **else**
12: **if** $k_{0x_j^{n+1}}(j - 1) = \Delta$ **then**
13: $c = \text{ins1_correct1}(n, a, y, j, h_1)$
14: **else**
15: $j \leftarrow j + 1$
16: **end if**
17: **end if**
18: **else**
19: **if** $k_{0x_j^{n+1}}(j - 1) + 2(n - j + 1) + 1 = \Delta$ **then**
20: $c = \text{ins1_correct1}(n, a, y, j, h_1)$
21: **else**
22: $j \leftarrow j + 1$
23: **end if**
24: **end if**
25: **end while**

Function 3: $c = \text{ins1_correct1}(n, a, y, j, h_1)$

Input: n, a, y, j, h_1 .
Output: $c = (c_1, c_2, \dots, c_n) \in C(n, 4)$.
1: **if** $c_j^{n+1}(j) = h_1$ **then**
2: $\text{Syn_new} = \sum_{i=1}^{j+1} i c_{j,i}^{n-1} + \sum_{i=j+1}^{n+1} (i - 1) c_{j,i}^{n+1} \text{ mod } (8n + 1)$
3: **else**
4: $j \leftarrow j + 1$
5: **end if**
6: **if** $\text{Syn_new} = a$ **then**
7: $c = (c_1, c_2, \dots, c_{j-1}, c_j, c_{j+1}, \dots, c_n)$
8: **else**
9: $j \leftarrow j + 1$
10: **end if**

Algorithm A1 finds the possible position j of the inserted symbol as steps 6, 13, 19 then uses Function 3 which presents function **ins1_correct1** to check this value of j to satisfy the constraint (12).

To determine the position of the inserted symbol, the value of Syn_new in line 2 of Function 3 indicates the syndrome of the received quaternary sequence c_j^{n+1} when not considering the inserted symbol $c_j^{n+1}(j)$. Thus, in the second term of the right-hand side, the coefficient needs to be $(i - 1)$. This syndrome is compared to the constraint (12) to obtain the position of the inserted symbols and finally, remove the inserted symbol in the

j -th position in c_j^{n+1} . Therefore, the quaternary sequence satisfies constraints (11), (12), and (13) can correct any one insertion error.

Appendix B. Two Consecutive Insertion Errors Correction

To correct the quaternary sequence when two consecutive insertion errors occur, we provide the details of correction procedure in Algorithm A2 and Function 4.

Algorithm A2 is constructed based on the analysis which is mentioned in Section 3.5. From steps 6, 13, 24, 31, the possible positions j and $j + 1$ of two consecutive insertion errors in the related binary sequence x can be obtained. However, since mapping (10) is used to map from quaternary symbols to binary bits, there can exist different cases of quaternary symbols which are mapped to the same binary bits, so we need to verify the exact quaternary values corresponding to the j -th and $(j + 1)$ -th positions.

Function 4: $c = \text{ins2_correct2}(j, n, a, y, h_1 + h_2)$

Input: $j, n, a, y, h_1 + h_2$.

Output: $c = (c_1, c_2, \dots, c_n) \in C(n, 4)$.

```

1: if  $c_j^{n+2}(j) = h_1$  and  $c_j^{n+2}(j + 1) = h_1 + h_2 - h_1$  then
2:    $Syn\_new = \sum_{i=1}^{j-1} i c_{j,i}^{n+2} + \sum_{i=j+2}^{n+2} (i - 2) c_{j,i}^{n+2} \bmod (8n + 1)$ 
3: else
4:    $j \leftarrow j + 1$ 
5: end if
6: if  $Syn\_new = a$  then
7:    $c = (c_1, c_2, \dots, c_{j-1}, c_j, c_{j+1}, \dots, c_n)$ 
8: else
9:    $j \leftarrow j + 1$ 
10: end if

```

The function **ins2_correct2** in Algorithm A2 is provided as Function 4 to output the unique sequence which satisfies three constraints (11), (12), and (13). Steps 1,2 in Function 4 correspond to the comparison to the constraints (13) and (12), respectively. This comparison determines the exact value and position of the inserted symbols as mentioned in Section 3.1.

In the similar way to Function 3 in Appendix A, the function Syn_new calculates syndrome of c_j^{n+2} when not considering the symbols in the j -th and $(j + 1)$ -th positions. This leads to the coefficient in the second term of function Syn_new is $(i - 2)$, meaning that the symbols which are after the $(j + 1)$ -th symbols are shifted to the left by 2 positions. The syndrome value Syn_new is compared to the value a of the constraint (12) to determine the exact values and positions of the inserted symbols of sequence. Obviously, the output sequence will satisfy both three constraints in Definition 4. Finally, two consecutive inserted symbols at the j -th and $(j + 1)$ -th positions of c_j^{n+2} are removed. The output of Function 4 is the corrected quaternary sequence.

Algorithm A2 Correct two consecutive insertion symbols

Input: $n, a, y, 0x_j^{n+2}$.
Output: $c = (c_1, c_2, \dots, c_n) \in C(n, 4)$.

- 1: Calculate Δ and $h_1 + h_2$ as in Table 3.
- 2: Set $j = 1$.
- 3: **if** $\Delta \geq 2\|r''\|$ **then**
- 4: **while** $j \leq n + 1$ **do**
- 5: **if** $\text{mod}(\Delta, 2) = 1$ **then**
- 6: **if** $2k_{0x_j^{n+2}}(j - 1) + 2(n - j) + 5 = \Delta$ **then**
- 7: $x_j = \bar{x}_{j-1}; x_{j+1} = x_{j-1}$
- 8: $c = \text{ins2_correct2}(j, n, a, y, h_1 + h_2)$
- 9: **else**
- 10: $j \leftarrow j + 1$
- 11: **end if**
- 12: **else**
- 13: **if** $2k_{0x_j^{n+2}}(j - 1) + 2(n - j) + 4 = \Delta$ **then**
- 14: $x_j = \bar{x}_{j-1}; x_{j+1} = \bar{x}_{j-1}$
- 15: $c = \text{ins2_correct2}(j, n, a, y, h_1 + h_2)$
- 16: **else**
- 17: $j \leftarrow j + 1$
- 18: **end if**
- 19: **end if**
- 20: **end while**
- 21: **else**
- 22: **while** $j \leq n + 1$ **do**
- 23: **if** $\text{mod}(\Delta, 2) = 1$ **then**
- 24: **if** $2k_{0x_j^{n+2}}(j - 1) + 1 = \Delta$ **then**
- 25: $x_j = x_{j-1}; x_{j+1} = \bar{x}_{j-1}$
- 26: $c = \text{ins2_correct2}(j, n, a, y, h_1 + h_2)$
- 27: **else**
- 28: $j \leftarrow j + 1$
- 29: **end if**
- 30: **else**
- 31: **if** $2k_{0x_j^{n+2}}(j - 1) = \Delta$ **then**
- 32: $x_j = x_{j-1}; x_{j+1} = x_{j-1}$
- 33: $c = \text{ins2_correct2}(j, n, a, y, h_1 + h_2)$
- 34: **else**
- 35: $j \leftarrow j + 1$
- 36: **end if**
- 37: **end if**
- 38: **end while**
- 39: **end if**

References

1. Goldman, N.; Bertone, P.; Chen, S.; Dessimoz, C.; LeProust, E.M.; Sipos, B.; Birney, E. Towards practical, high-capacity, low-maintenance information storage in synthesized DNA. *Nature* **2013**, *494*, 77–80. [[CrossRef](#)] [[PubMed](#)]
2. Blawat, M.; Gaedke, K.; Hütter, I.; Chen, X.; Turczyk, B.; Inverso, S.; Pruitt, B.; Church, G. Forward error correction for DNA data storage. *Procedia Comput. Sci.* **2016**, *80*, 1011–1022. [[CrossRef](#)]
3. Erlich, Y.; Zielinski, D. DNA Fountain enables a robust and efficient storage architecture. *Science* **2016**, *355*, 950–954. [[CrossRef](#)] [[PubMed](#)]
4. Heckel, R.; Mikutis, G.; Grass, R. A characterization of the DNA data storage channel. *Sci. Rep.* **2019**, *9*, 1–12.
5. Varshamov, R.; Tenengolts, G. A code that corrects single asymmetric errors. *Autom. Telemekhanika* **1965**, *26*, 288–292.
6. Levenshtein, V.I. Binary codes capable of correcting deletions, insertions, and reversals. *Sov. Phys. Dokl.* **1966**, *10*, 707–710.

7. Levenshtein, V.I. Asymptotically optimum binary codes with correction for losses of one or two adjacent bits. *Syst. Theo. Res.* **1970**, *19*, 298–304.
8. Cheng, L.; Swart, T.; Ferreira, H.; Abdel-Ghaffar, K. Codes for correcting three or more consecutive deletions or insertions. In Proceedings of the 2014 IEEE International Symposium on Information Theory, Honolulu, HI, USA, 29 June–4 July 2014; pp. 1246–1250.
9. Schoeny, C.; Wachter-Zeh, A.; Gabrys, R.; Yaakobi, E. Codes correcting a burst of deletions or insertions. *IEEE Trans. Inf. Theory* **2017**, *63*, 1971–1985. [[CrossRef](#)]
10. Chee, Y.; Kiah, H.; Nguyen, T. Linear-time encoders for codes correcting a single edit for DNA-based data storage. In Proceedings of the 2019 IEEE International Symposium on Information Theory (ISIT), Paris, France, 7–12 September 2019; pp. 773–776.
11. Nguyen, T.; Cai, K.; Immink, K.; Kiah, H. Capacity-approaching constrained codes with error correction for DNA-based data storage. *IEEE Trans. Inf. Theory* **2021**, *67*, 5602–5613. [[CrossRef](#)]
12. Bornholt, J.; Lopez, R.; Carmean, D.; Ceze, L.; Seelig, G. A DNA-based archival storage system. In Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems, Atlanta, GA, USA, 2–6 April 2016; pp. 637–649.
13. Ross, M.; Russ, C.; Costello, M.; Hollinger, A.; Lennon, N.; Hegarty, R.; Nusbaum, C.; Jaffe, D. Characterizing and measuring bias in sequence data. *Genome Bio.* **2013**, *14*, R51. [[CrossRef](#)] [[PubMed](#)]
14. Cai, K.; Chee, Y.; Gabrys, R.; Kiah, H.; Nguyen, T. Correcting a single indel/edit for DNA-based data storage: Linear-time encoders and order-optimality. *IEEE Trans. Inf. Theory* **2021**, *67*, 3438–3451. [[CrossRef](#)]
15. Tenengolts, G. Nonbinary codes, correcting single deletion or insertion. *IEEE Trans. Inf. Theory* **1984**, *30*, 766–769. [[CrossRef](#)]
16. Schoeny, C.; Sala, F.; Dolecek, L. Novel combinatorial coding results for DNA sequencing and data storage. In Proceedings of the 2017 51st Asilomar Conf. Signals, Systems, and Computers, Pacific Grove, CA, USA, 29 October–1 November 2017; pp. 511–515.
17. Paluni, F.; Swart, T.; Weber, J.; Ferreira, H.; Clarke, W. A note on non-binary multiple insertion/deletion correcting codes. In Proceedings of the 2011 IEEE Information Theory Workshop, Paraty, Brazil, 16–20 October 2011; pp. 683–687.
18. Sima, J.; Raviv, N.; Bruck, J. Two deletion correcting codes from indicator vectors. *IEEE Trans. Inf. Theory* **2020**, *66*, 2375–2391. [[CrossRef](#)]
19. Sima, J.; Gabrys, R.; Bruck, J. Optimal codes for the q-ary deletion channel. In Proceedings of the 2020 IEEE International Symposium on Information Theory (ISIT), Los Angeles, CA, USA, 21–26 June 2020; pp. 740–745.
20. Sima, J.; Gabrys, R.; Bruck, J. Optimal systematic t-deletion correcting codes. In Proceedings of the 2020 IEEE International Symposium on Information Theory (ISIT), Los Angeles, CA, USA, 21–26 June 2020; pp. 769–774.
21. Sima, J.; Bruck, J. On optimal k-deletion correcting codes. *IEEE Trans. Inf. Theory* **2020**, *67*, 3360–3375. [[CrossRef](#)]
22. Wang, S.; Sima, J.; Farnoud, F. Non-binary codes for correcting a burst of at most 2 deletions. In Proceedings of the 2021 IEEE International Symposium on Information Theory (ISIT), Melbourne, Australia, 12–20 July 2021; pp. 2804–2809.
23. No, A. Nonasymptotic upper bounds on binary single deletion codes via mixed integer linear programming. *Entropy* **2019**, *21*, 1202. [[CrossRef](#)]
24. Immink, K.; Cai, K. Properties and constructions of constrained codes for DNA-based data storage. *IEEE Access* **2020**, *8*, 49523–49531. [[CrossRef](#)]