Coding for Racetrack Memories

Yeow Meng Chee, *Senior Member, IEEE*, Han Mao Kiah, Alexander Vardy, *Fellow, IEEE*, Van Khu Vu, and Eitan Yaakobi, *Senior Member, IEEE*

***Abstract*— *Racetrack memory* is a new technology, which utilizes magnetic domains along a nanoscopic wire in order to obtain extremely high storage density. In racetrack memory, each magnetic domain can store a single bit of information, which can be sensed by a reading *port* (*head*). The memory is structured like a tape, which supports a *shift* operation that moves the domains to be read sequentially by the head. In order to increase the memory's speed, prior work studied how to minimize the latency of the shift operation, while the no less important reliability of this operation has received only a little attention. In this paper, we design codes, which combat shift errors in racetrack memory, called *position errors*, namely, shifting the domains is not an error-free operation and the domains may be over shifted or are not shifted, which can be modeled as *deletions* and *sticky insertions*. While it is possible to use conventional deletion and insertion-correcting codes, we tackle this problem with the special structure of racetrack memory, where the domains can be read by multiple heads. Each head outputs a noisy version of the stored data and the multiple outputs are combined in order to reconstruct the data. This setup is a special case of the *reconstruction problem* studied by Levenshtein, however, in our case, the position errors from different heads are correlated. We will show how to take advantage of this special feature of racetrack memories in order to construct codes correcting deletions and sticky insertions. In particular, under this paradigm, we will show that it is possible to correct, with at most a single bit of redundancy, d deletions with $d+1$ heads if the heads are well separated. Similar results are provided for burst of deletions, sticky insertions, and combinations of both deletions and sticky insertions.**

***Index Terms*— Racetrack memory, deletions, sticky insertions, run-length limited constrained codes.**

Manuscript received June 5, 2017; revised December 25, 2017 and January 22, 2018; accepted February 6, 2018. Date of publication February 19, 2018; date of current version October 18, 2018. Y. M. Chee was supported by the Singapore Ministry of Education under Grant MOE2015-T2-2-086. H. M. Kiah was supported in part by the Singapore Ministry of Education under Grant MOE2015-T2-2-086 and in part by the Singapore Ministry of Education under Grant MOE2016-T1-001-156. A. Vardy was supported by the National Science Foundation under Grants CCF–1405119 and CCF–1719139. E. Yaakobi was supported by the Israel Science Foundation under Grant 1624/14. This paper was presented in part at the 2017 IEEE International Symposium on Information Theory [3].

Y. M. Chee, H. M. Kiah, and V. K. Vu are with the School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore 637371 (e-mail: ymchee@ntu.edu.sg; hmkiah@ntu.edu.sg; vankhu001@ntu.edu.sg).

A. Vardy is with the School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore 637371, also with the Department of Electrical and Computer Engineering, Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA 92093 USA, and also with the Department of Mathematics, University of California at San Diego, La Jolla, CA 92093 USA (e-mail: avardy@ucsd.edu).

E. Yaakobi is with the Department of Computer Science, Technion—Israel Institute of Technology, Haifa 3200003, Israel (e-mail: yaakobi@cs.technion.ac.il).

Communicated by V. Sidorenko, Associate Editor for Coding Theory.
Digital Object Identifier 10.1109/TIT.2018.2807480

I. Introduction

RACETRACK memory, also known as *domain wall memory*, is an emerging non-volatile memory which is based on spintronic technology. It attracts significant attention due to its promising ultra-high storage density, even comparing to other spintronic memory technologies such as STT-RAM [32].

A racetrack memory is composed of *cells*, also called *domains*, which are positioned on a tape-like stripe and are separated by *domain walls*. The magnetization of a domain is programmed to store a single bit value, which can be read by sensing its magnetization direction. The reading mechanism is operated by a read-only *port*, called a *head*, together with a *reference domain*. Since the head is fixed (i.e., cannot move), a *shift* operation is required in order to read all the domains. Shifting the cells is accomplished by applying shift current which moves the domain walls in one direction. Thus, shift operations move all the domains one step either to the right or to the left. It is also possible to shift by more than a single step by applying a stronger current. When doing so, it is required to have more than a single head to read the domain walls [19], [26], [32]. Multiple heads can also be used in order to significantly reduce the read access latency of the memory. Usually these heads are distributed along the racetrack uniformly in order to reduce the maximum shift distance. However, since multiple heads incur a price of area overhead, it is still desirable not to add a large number of heads [26], [32].

There are several approaches to enhance the shift operation in order to reduce its time and energy consumption [24], [26]. However these mechanisms suffer from degraded reliability and cannot ensure that domains are perfectly shifted so they are aligned with the head. These errors, called *position errors*, can be modeled as deletions and sticky insertions [32], which is the motivation for this work. A deletion is the event where the domains are shifted by more than a single domain location and thus one of the domains is not read, which results in a *deletion* of the bit stored in this domain. In case the domains were not successfully shifted, then the same domain is read again and we experience an *insertion*, however of the same bit. This kind of insertion errors is also referred as *repetition errors* [6] or *sticky insertions* in a *sticky channel* [6], [17], [18].

In this work we study codes which correct position errors in racetrack memory. At a first sight, this problem is not any different than the well-studied problem of designing codes correcting deletions and insertions [1], [8], [11], [14].

However, we take here another approach to tackle the problem and leverage the special features of racetrack memory, where it is possible to use more than a single head in order to read the domains. Thus, each domain is read more than once and the extra reads can be used in order to correct the position errors during the read process. Since every head reads all the bits, we can treat every head as a channel which returns a noisy version of the stored information, and based on these noisy reads the information is decoded. This model falls under the general framework by Levenshtein of the *reconstruction problem* [12], [13]. However, in our case, as opposed to the general one studied by Levenshtein, the position errors are correlated and depend upon the locations and distance between the heads. Another similar model where bits are read more than once and the errors are correlated is the *symbol-pair channel*, where every pair of adjacent bits is read together, see [2], [4], [5], [31].

In contrast to substitution errors, deletions/sticky insertions behave *differentially*. Namely, to successfully decode a substitution error, it is necessary to determine the location of the error. However, for deletions/sticky insertions, the decoder can successfully decode the correct codeword without determining all the locations of the deletions/sticky insertions, since it could be any bit which belongs to the run where each deletion/sticky insertion has occurred. Assume first that the heads are adjacent and on every cycle the domains are shifted by a single location. Thus, if there are no position errors, the bit stored in each domain is read twice. On the other hand, in the occurrence of position errors, the deletions/sticky insertions in the two heads are correlated. For example, if the ith bit is deleted in the first head then the $(i+1)$-st bit is deleted in the second head. In case these two deleted bits belong to the same run, then the noisy words from the two heads are identical and thus we did not benefit from the extra read by the additional head. On the other hand, if the heads are well separated and there are no long runs in the stored information, then the heads' outputs will differ and under this setup we will show how it is possible to correct the position errors. Note that it is possible to correct a fixed number of deletions and sticky insertions with a single head while the rate of the codes approaches 1 and the redundancy order is $\Theta(\log(n))$ [1], [11], where n is the code length. Hence, any code construction using multiple heads should have rate approaching 1 and more than that, improve upon the redundancy result of $\Theta(\log(n))$. However, this should be accomplished while minimizing the distance between the heads.

The rest of this paper is organized as follows. In Section II, we formally define the model and problems studied in the paper, namely the reading process in racetrack memory and codes correcting deletions and sticky insertions using multiple heads. In Section III, we construct codes correcting a single deletion using two heads with approximately 0.36 redundancy bits, by requiring the distance between the heads to be at least $\lceil \log(n) \rceil + 1$. In Section IV, we extend this construction for codes correcting a burst of deletions while the length of the burst is either exactly b or at most b. In the former case the redundancy is again approximately 0.36 bits but the distance between the heads should be at least $\lceil \log(n) \rceil + b$, while in the latter the code redundancy is at most a single bit and the heads are located at least $\lceil \log(n) \rceil + b + 1$ positions apart. Another extension is given in Section V for codes correcting multiple deletions. In this case our construction can correct d deletions using $d+1$ heads with at most a single bit of redundancy, while the distance between adjacent heads is at least $d\lceil \log(n) \rceil + d(d+1)/2 + 1$. In case the number of heads m is strictly less than $d+1$, we show that it is possible to correct $m-1$ deletions with the heads, and so the code should be able to correct the remainder of $d-(m-1)$ deletions. In Section VI, we study codes correcting sticky insertions and in Section VII, we study codes correcting both deletions and sticky insertions. We also extend our construction for codes correcting a combination of substitutions and deletions (or sticky insertions) in Section VIII. Finally, we discuss some related problems and extensions in Section IX and conclude the paper in Section X.

II. Preliminaries and Model Definitions

Let $\mathbb{F}_2$ denote the binary finite field. For a positive integer n, the set $\{1, 2, \ldots, n\}$ is denoted by $[n]$. A *binary word* of length n over the alphabet $\mathbb{F}_2$ is a vector in $\mathbb{F}_2^n$. For $i \in [n]$, the ith coordinate of a word $\boldsymbol{u} \in \mathbb{F}_2^n$ is denoted by u_i, so that $\boldsymbol{u} = (u_1, u_2, \ldots, u_n)$. *A binary code* of length n is a subset $\mathbb{C} \in \mathbb{F}_2^n$. Each element of $\mathbb{C}$ is called *a codeword*. For each code $\mathbb{C}$ of length n, we define the *rate* of the code $\mathbb{C}$ to be $R(\mathbb{C}) = \log(|\mathbb{C}|)/n$ and the *redundancy* of the code $\mathbb{C}$ to be $r(\mathbb{C}) = n - \log(|\mathbb{C}|)$ where $|\mathbb{C}|$ is the size of the code $\mathbb{C}$.

Let $\boldsymbol{u} = (u_1, \ldots, u_n)$ and $\boldsymbol{v} = (v_1, \ldots, v_m)$ be two vectors of length n and m, respectively. The concatenation of $\boldsymbol{u}$ and $\boldsymbol{v}$ is the vector $(u_1, \ldots, u_n, v_1, \ldots, v_m)$ of length $n+m$, which is denoted by $\boldsymbol{u} \circ \boldsymbol{v}$. A subvector of a word $\boldsymbol{u}$ is a vector $\boldsymbol{u}[i_1, i_2] = (u_{i_1}, u_{i_1+1}, \ldots, u_{i_2})$ in which $1 \leqslant i_1 \leqslant i_2 \leqslant n$. The length of this subvector is $1 \leqslant i_2 - i_1 + 1 \leqslant n$. In case $i_1 = i_2 = i$, we denote a subvector $\boldsymbol{u}[i, i]$ of length 1 by $\boldsymbol{u}[i]$ to specify the i-th element of vector $\boldsymbol{u}$.

Let ℓ and m be two positive integers where $\ell \leqslant m$. Then, a length-m vector $\boldsymbol{v} \in \mathbb{F}_2^m$ which satisfies $v_i = v_{i+\ell}$ for all $1 \leqslant i \leqslant m - \ell$ is said to have *period* ℓ. For a vector $\boldsymbol{u} \in \mathbb{F}_2^n$, we denote by $L(\boldsymbol{u}, \ell)$ the length of its longest subvector which has period ℓ. Note that by definition $L(\boldsymbol{u}, \ell) \geqslant \ell$, and for $\ell = 1$, $L(\boldsymbol{u}, 1)$ equals the length of the longest run in $\boldsymbol{u}$.

Example 1: Let $\boldsymbol{u} = (u_1, \ldots, u_9) = (0, 0, 1, 1, 0, 1, 0, 1, 1) \in \mathbb{F}_2^9$ be a word of length 9. Since the longest run in $\boldsymbol{u}$ is of length two, we have $L(\boldsymbol{u}, 1) = 2$. The subvector $\boldsymbol{u}[4, 8] = (1, 0, 1, 0, 1)$ of $\boldsymbol{u}$ has period 2 since $u_4 = u_6 = u_8 = 1$ and $u_5 = u_7 = 0$. This is the longest subvector of $\boldsymbol{u}$ of period 2, and hence $L(\boldsymbol{u}, 2) = 5$. □

For a length-n word $\boldsymbol{u} \in \mathbb{F}_2^n$ and $i \in [n]$, we denote by $\boldsymbol{u}(\delta_i)$ the vector obtained by $\boldsymbol{u}$ after deleting its ith bit, that is, $\boldsymbol{u}(\delta_i) = (u_1, \ldots, u_{i-1}, u_{i+1}, \ldots, u_n)$. For a set $\Delta \subseteq \{\delta_i : i \in [n]\}$, we denote by $\boldsymbol{u}(\Delta)$ the vector of length $n - |\Delta|$ obtained from $\boldsymbol{u}$ after deleting all the bits specified by the locations in the set Δ. In case $\Delta = \{\delta_i, \ldots, \delta_{i+b-1}\}$ then we denote the vector $\boldsymbol{u}(\Delta)$ by $\boldsymbol{u}(\delta_{[i,b]})$ to specify a burst of b deletions starting at the ith position.

Example 2: Let $\boldsymbol{u} = (0, 0, 1, 1, 0, 1, 0, 1, 1) \in \mathbb{F}_2^9$, then $\boldsymbol{u}(\delta_4) = (0, 0, 1, 0, 1, 0, 1, 1)$. For $\Delta = \{\delta_4, \delta_7, \delta_9\}$ then $\boldsymbol{u}(\Delta) = (0, 0, 1, 0, 1, 1)$, and $\boldsymbol{u}(\delta_{[3,4]}) = (0, 0, 0, 1, 1)$. □

Fig. 1. Racetrack memory with multiple heads

In this work, we assume that the information stored in the racetrack memory is represented by a word $\boldsymbol{u}$. The memory is comprised of magnetizable cells which can store a single bit. The information is read back from the cells by sensing their magnetization direction using heads which are fixed in their positions; see Fig. 1.

Since the heads are fixed in their locations, the memory cells move so they can all be read by the heads. This *shifting operation* is performed by applying a shift current which moves all the cells on each cycle one or more steps in the same direction. However, the shifting mechanism does not work perfectly and may suffer from errors, called *position errors*. That is, cells may be shifted by more than a single location on each cycle or are not shifted. These position errors can be modeled as deletions and sticky insertions. Namely, a *single deletion* is the event where the cells are shifted by two locations instead of one and thus one of the bits is not read by the head. In case the cells were shifted by some $b+1 > 2$ locations, then b consecutive cells were not read and we say that a *deletion burst of size b* has occurred. On the other hand, a *sticky insertion* is the event where the cells were *not* shifted and the same cell is read again and if this happens $b > 1$ times in a row, we say that a *burst of b sticky insertions* has occurred. The goal of this work is to construct codes for racetrack memories which aim to correct this class of position errors.

In this work we assume that there are several heads and each head reads all the cells. In case there is only a single head, then the only approach to correct the position errors is by using a code which is capable of correcting deletions and sticky insertions. However, in case there are several heads, the cells are read multiple times by each head and thus we study how this inherent redundancy can be used to design better codes. The output of the heads depend on their locations. For example, assume that there are three heads which are used to read the stored word $\boldsymbol{u}$. Assume also that the distance between the first two heads is t_1 and the distance between the last two heads is t_2. Then, if a deletion occurs at position i in the first head then a deletion also occurs at position $i + t_1$ in the second head and another deletion at position $i+t_1+t_2$ in the third head. Therefore, the output of the first, second, third head is the vector $\boldsymbol{u}(\delta_i), \boldsymbol{u}(\delta_{i+t_1}), \boldsymbol{u}(\delta_{i+t_1+t_2})$, respectively. A specific scenario of this setup is given in the next example.

Example 3: Let $\boldsymbol{u} = (0, 0, 1, 1, 0, 1, 0, 1, 1) \in \mathbb{F}_2^9$ be the word stored in the memory, and assume that there are three heads which are positioned with $t_1 = 1$ positions between the first and second heads and $t_2 = 2$ positions between the second and third heads. Assume that a deletion occurs at position 3 in the first head, then a deletion also occurs at position 4 in the second head and at position 6 in the third head. Hence, the outputs from the three heads are:

$$\textbf{Head 1}: \boldsymbol{u}(\delta_3) = (0, 0, 1, 0, 1, 0, 1, 1)$$
$$\textbf{Head 2}: \boldsymbol{u}(\delta_4) = (0, 0, 1, 0, 1, 0, 1, 1)$$
$$\textbf{Head 3}: \boldsymbol{u}(\delta_6) = (0, 0, 1, 1, 0, 0, 1, 1).$$

□

As illustrated in Example 3, different heads may have the same output if the distance between their locations is small and the stored word has a long run. However, if the heads are well separated then their outputs is more likely to be different.

The goal in this paper is to design codes which can correct position errors in the reading process. We say that a code is an *m-head b-position-error-correcting code* if it can correct b position errors using m heads. Similarly, we also define *m-head b-deletion-correcting codes, m-head b-sticky-insertion-correcting codes*, *m-head b-burst-deletion-correcting codes*, and *m-head b-burst-sticky-insertion-correcting codes*. We note that the locations of the heads can also be part of the code design, however they should not be too far apart from each other since the area for shifting the cells may also be constrained and thus should be minimized. As always, the goal in designing these codes is to minimize the redundancy of each code construction.

III. Two-Head Single-Deletion-Correcting Codes

In this section we study how to construct two-head single-deletion-correcting codes. Our main result states that if the distance between the two heads is at least $\lceil \log(n) \rceil + 1$ cells, where n is the length of the codewords, then such codes exist with redundancy of roughly 0.36 bits. Let us remind that if the vector $\boldsymbol{u} = (u_1, \ldots, u_n)$ is stored in the memory and the distance between the two heads is t locations, then if a deletion of the ith bit occurs at the first head, its output is $\boldsymbol{u}(\delta_i)$ while the output of the second head is $\boldsymbol{u}(\delta_{i+t})$.

We are now ready to present the construction of double-head single-deletion-correcting codes.

Construction 1: For all $t \leqslant n$, let $\mathbb{C}_1(n, 1, t)$ be a code of length n such that the length of the longest run of every codeword is at most t. That is, $\mathbb{C}_1(n, 1, t) = \{\boldsymbol{c} \in \mathbb{F}_2^n \mid L(\boldsymbol{c}, 1) \leqslant t\}$.

The correctness of this construction is proved in the next theorem.

Theorem 2: The code $\mathbb{C}_1(n, 1, t)$ is a two-head single-deletion-correcting code when the heads are positioned t locations apart.

Proof: Let $\boldsymbol{c} = (c_1, \ldots, c_n) \in \mathbb{C}_1(n, 1, t)$ be a stored codeword of length n and assume that a single deletion occurred at position i. Then, the outputs from the two heads are:

$$\textbf{Head 1}: \boldsymbol{c}(\delta_i) = (c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_n),$$
$$\textbf{Head 2}: \boldsymbol{c}(\delta_{i+t}) = (c_1, \ldots, c_{i+t-1}, c_{i+t+1}, \ldots, c_n).$$

Consider the first $i + t - 1$ bits in these two sequences:

$$\textbf{Head 1}: \boldsymbol{c}(\delta_i)[1, i + t - 1] = (c_1, \ldots, c_{i-1}, c_{i+1}, c_{i+2}, \ldots, c_{i+t}),$$
$$\textbf{Head 2}: \boldsymbol{c}(\delta_{i+t})[1, i + t - 1] = (c_1, \ldots, c_{i-1}, c_i, c_{i+1}, \ldots, c_{i+t-1}).$$

We claim that $\boldsymbol{c}(\delta_i)[1, i + t - 1] \neq \boldsymbol{c}(\delta_{i+t})[1, i + t - 1]$. Otherwise, we will get that

$$c_i = c_{i+1} = \cdots = c_{i+t-1} = c_{i+t},$$

which implies that there is a run of length $t + 1$ in $\boldsymbol{c}$ in contradiction to the construction of the code $\mathbb{C}_1(n, 1, t)$. Let j_1 be the leftmost index that differs between $\boldsymbol{c}(\delta_i)[1, i + t - 1]$ and $\boldsymbol{c}(\delta_{i+t})[1, i+t-1]$. Such an index exists since $\boldsymbol{c}(\delta_i)[1, i + t - 1] \neq \boldsymbol{c}(\delta_{i+t})[1, i + t - 1]$ and so $j_1 \leqslant i + t - 1$. Furthermore, $j_1 \geqslant i$ since the first $i - 1$ bits in the outputs from the two heads are the same as in the stored codeword. Note that j_1 can be different from i in case the ith bit which was deleted is in a middle of a run and so the first occurrence where $\boldsymbol{c}(\delta_i)[1, i + t - 1]$ and $\boldsymbol{c}(\delta_{i+t})[1, i + t - 1]$ differ is only at the end of this run. We conclude that $\boldsymbol{c}[1, j_1] = \boldsymbol{c}(\delta_{i+t})[1, j_1]$ and $\boldsymbol{c}[j_1 + 1, n] = \boldsymbol{c}(\delta_i)[j_1, n - 1]$. Hence, the original codeword $\boldsymbol{c}$ can be recovered by concatenating the first j_1 bits from $\boldsymbol{c}(\delta_{i+t})$ and the last $n - j_1$ bits from $\boldsymbol{c}(\delta_i)$. That is, $\boldsymbol{c} = \boldsymbol{c}(\delta_{i+t})[1, j_1] \circ \boldsymbol{c}(\delta_i)[j_1, n-1]$. This proof provides also a simple decoding algorithm for the code $\mathbb{C}_1(n, 1, t)$. ■

The next example demonstrates this code construction and its decoder.

Example 4: Let $n = 9, t = 3$ and $\boldsymbol{c} = (0, 0, 1, 1, 0, 1, 0, 1, 1)$ be a stored codeword in $\mathbb{C}_1(n, 1, t)$. Let us assume that the outputs from the two heads are:

$$\textbf{Head 1}: \boldsymbol{c}(\delta_3) = (0, 0, 1, 0, 1, 0, 1, 1),$$
$$\textbf{Head 2}: \boldsymbol{c}(\delta_6) = (0, 0, 1, 1, 0, 0, 1, 1).$$

Hence $j_1 = 4$ is the leftmost index that differs between the two vectors, and the stored codeword is $\boldsymbol{c} = \boldsymbol{c}(\delta_6)[1, 4] \circ \boldsymbol{c}(\delta_3)[4, 8] = (0, 0, 1, 1, 0, 1, 0, 1, 1)$. □

By a suitable mapping described in Section IV, the code $\mathbb{C}_1(n, 1, t)$ can be transformed into a code that satisfies the $(0, t-1)$ *Run Length Limited* (*RLL*) constraint [9], [29]. While for each fixed t efficient encoding and decoding algorithms are known for codes which satisfy the $(0, t - 1)$ RLL constraint, the rates of these codes is strictly less than 1. Since we can achieve codes with rate 1 by simply using a single head and a single-deletion-correcting code of redundancy at most $\log(n+1)$, we are interested only in codes with rate 1 and will optimize their redundancy. Thus, we follow a similar approach to the one taken in [23] for codes correcting bursts of deletions and let t be a function of the code length n. In particular, by choosing $t = \lceil \log(n) \rceil + 1$, it was observed in [23], using the derivations from [21] and [22], that the redundancy of the code $\mathbb{C}_1(n, 1, \lceil \log(n) \rceil + 1)$ is approximately $\log(e)/4 \approx 0.36$. Recently, in [15] this result was proved where the idea was to use existing results from constrained codes in order to give tight upper and lower bounds on the size of the code $\mathbb{C}_1(n, 1, \lceil \log(n) \rceil + 1)$ such that its asymptotic redundancy converges to 0.36. Furthermore, for $t = \lceil \log(n) \rceil + 2$ efficient encoding and decoding algorithms were recently found for these codes using a single bit of redundancy [15]. We conclude this discussion with the following corollary.

Corollary 3: There exists a two-head single-deletion-correcting code when the heads are positioned $t = \lceil \log(n) \rceil + 1$ locations apart with redundancy of approximately $\log(e)/4 \approx 0.36$ bits.

IV. Codes Correcting a Burst of Deletions

In this section we study the setup where the domains are over-shifted by more than a single location and thus a burst of deletions occurs in each head. More specifically, we study the construction of two-head b-burst-deletion-correcting codes. We will focus on two cases: the length of the burst is exactly b or at most b.

A. Two-Head b-Burst-Deletion-Correcting Codes

Here we investigate codes correcting a burst of exactly b adjacent deletions using two heads. Suppose we use two heads at distance t to correct a burst of size b in the stored codeword $\boldsymbol{c} = (c_1, \ldots, c_n)$. Recall that for $i \in [n]$ and $b \in [n - i]$, the vector obtained from $\boldsymbol{c}$ after deleting the subvector $c[i, i + b - 1] = (c_i, \ldots, c_{i+b-1})$ is $\boldsymbol{c}(\delta_{[i,b]})$. Therefore, we know that if the output from the first head is $\boldsymbol{c}(\delta_{[i,b]})$ for some i and b, then the output from the second head is $\boldsymbol{c}(\delta_{[i+t,b]})$, where the heads are located t positions apart. The following is the construction of such codes.

Construction 4: Let $\mathbb{C}_2(n, b, t)$ be a code of length n such that the length of the longest subvector which has period b of every codeword $\boldsymbol{c} \in \mathbb{C}_2(n, b, t)$ is at most t. That is, $\mathbb{C}_2(n, b, t) = \{\boldsymbol{c} \in \mathbb{F}_2^n \mid L(\boldsymbol{c}, b) \leqslant t\}$.

The proof that this construction can correct a burst of deletion of length b follows similar ideas to that in the proof of Theorem 2.

Theorem 5: The code $\mathbb{C}_2(n, b, t)$ is a two-head b-burst-deletion-correcting code when the heads are positioned t locations apart.

Proof: Let $\boldsymbol{c} = (c_1, \ldots, c_n) \in \mathbb{C}_2(n, b, t)$ be a stored codeword and assume that a burst of b consecutive bits $\boldsymbol{c}[i, i + b - 1]$ is deleted in the first head, and thus the subvector $\boldsymbol{c}[i + t, i + t + b - 1]$ is deleted in the second head. Then, the outputs from the two heads are:

$$\textbf{Head 1}: \boldsymbol{c}(\delta_{[i,b]}) = (c_1, \ldots, c_{i-1}, c_{i+b}, \ldots, c_n)$$
$$\textbf{Head 2}: \boldsymbol{c}(\delta_{[i+t,b]}) = (c_1, \ldots, c_{i+t-1}, c_{i+t+b}, \ldots, c_n).$$

We claim that the first $i + t - b$ bits in those two vectors, i.e.,

$$\boldsymbol{c}(\delta_{[i,b]})[1, i+t-b] = (c_1, \ldots, c_{i-1}, c_{i+b}, c_{i+b+1}, \ldots, c_{i+t}),$$
$$\boldsymbol{c}(\delta_{[i+t,b]})[1, i+t-b] = (c_1, \ldots, c_{i-1}, c_i, c_{i+1}, \ldots, c_{i+t-b}),$$

are different. Assume the contrary that equality holds between the last two vectors. Then, we get that $c_j = c_{j+b}$ for $i \leqslant j \leqslant i + t - b$ and thus the subvector $\boldsymbol{c}[i, i + t]$ of length $t + 1$ has period b, in contradiction to the construction of the code $\mathbb{C}_2(n, b, t)$. Let j_1 be the leftmost index that differs between $\boldsymbol{c}(\delta_{[i,b]})[1, i + t - b]$ and $\boldsymbol{c}(\delta_{[i+t,b]})[1, i + t - b]$. Similarly to the proof of Theorem 2, such an index j_1 exists and $i \leqslant j_1 \leqslant i + t - b$. Thus, $i + b - 1 \leqslant j_1 + b - 1 \leqslant i + t - 1$. We note that the first $i + t - 1$ bits in $\boldsymbol{c}(\delta_{[i+t,b]})$ are correct, that is, $\boldsymbol{c}(\delta_{[i+t,b]})[1, i + t - 1] = \boldsymbol{c}[1, i + t - 1]$, and since $j_1 + b - 1 \leqslant i + t - 1$, we deduce that

$$\boldsymbol{c}[1, j_1 + b - 1] = \boldsymbol{c}(\delta_{[i+t,b]})[1, j_1 + b - 1]. \tag{1}$$

Following the same arguments, the last $n - i - b + 1$ bits in $\boldsymbol{c}(\delta_{[i,b]})$ are correct, that is, $\boldsymbol{c}(\delta_{[i,b]})[i, n - b] = \boldsymbol{c}[i + b, n]$. Since $j_1 + b \geqslant i + b$, we get that

$$\boldsymbol{c}[j_1 + b, n] = \boldsymbol{c}(\delta_{[i,b]})[j_1, n - b]. \tag{2}$$

Combining (1) and (2) we conclude that the stored codeword $\boldsymbol{c}$ can be recovered as follows:

$$\boldsymbol{c} = \boldsymbol{c}(\delta_{[i+t,b]})[1, j_1 + b - 1] \ \circ \ \boldsymbol{c}(\delta_{[i,b]})[j_1, n - b].$$

■

The next example demonstrates the decoding procedure in Theorem 5.

Example 5: Let $n = 10, b = 2, t = 3$ and

$$\boldsymbol{c} = (0, 0, 1, 1, 0, 1, 1, 0, 1, 1)$$

be a stored codeword in $\mathbb{C}(n, b, t)$. Assume that the two adjacent bits c_3 and c_4 are deleted in the first head, so the bits c_6 and c_7 are deleted in the second head. Then, the outputs from the two heads are:

$$\textbf{Head 1}: \boldsymbol{c}(\delta_{[3,2]}) = (0, 0, 0, 1, 1, 0, 1, 1),$$
$$\textbf{Head 2}: \boldsymbol{c}(\delta_{[6,2]}) = (0, 0, 1, 1, 0, 0, 1, 1).$$

Therefore, $j_1 = 3$ is the leftmost index that differs between the two vectors, and we conclude that the stored codeword is $\boldsymbol{c} = \boldsymbol{c}(\delta_{[6,2]})[1, 4] \circ \boldsymbol{c}(\delta_{[3,2]})[3, 8] = (0, 0, 1, 1, 0, 1, 1, 0, 1, 1)$. □

Next we turn to evaluate the size of the code $\mathbb{C}_2(n, b, t)$. In particular, as done in the previous section, we will find a value of t for which the redundancy of the code will be approximately 0.36 bits. We start with the following definition.

*Definition 6: Let $\boldsymbol{u} = (u_1, u_2, \ldots, u_m) \in \mathbb{F}_2^m$ be a length-m vector. For $b < m$, the b-*period check vector *of $\boldsymbol{u}$ is the vector $\boldsymbol{p}_b(\boldsymbol{u}) = (u_1 + u_{1+b}, u_2 + u_{2+b}, \ldots, u_{m-b} + u_m)$ of length $m - b$.*

The following lemma can be readily verified.

Lemma 7: A word $\boldsymbol{u}$ contains a subvector of length t with period b if and only if $\boldsymbol{p}_b(\boldsymbol{u})$ contains a run of $t - b$ zeroes.

For a vector $\boldsymbol{u}$, we denote by $L_0(\boldsymbol{u})$ the length of the longest run of zeroes in $\boldsymbol{u}$. For example $L_0(0110100010) = 3$. For all n and $t \leqslant n$, we define the code $\mathcal{R}(n, t)$ to be

$$\mathcal{R}(n, t) = \{\boldsymbol{c} \in \mathbb{F}_2^n \mid L_0(\boldsymbol{u}) \leqslant t\}.$$

Using Lemma 7, we can construct a bijection between $\mathbb{C}_2(n, b, t)$ and the set $\mathbb{F}_2^b \times \mathcal{R}(n - b, t - b)$ for $n \geqslant b + 1$. Specifically, we define the following maps.

- $\Phi : \mathbb{C}_2(n, b, t) \to \mathbb{F}_2^b \times \mathcal{R}(n - b, t - b)$, where $\Phi(\boldsymbol{u}) = (\boldsymbol{u}[1, b], \boldsymbol{p}_b(\boldsymbol{u}))$.
- $\Psi : \mathbb{F}_2^b \times \mathcal{R}(n-b, t-b) \to \mathbb{C}_2(n, b, t)$, where $\Psi(\boldsymbol{v}, \boldsymbol{w}) = \boldsymbol{u}$ and

$$u_i = \begin{cases} v_i, & \text{if } i \leqslant b, \\ u_{i-b} + w_{i-b}, & \text{otherwise.} \end{cases}$$

In the context of error-correcting codes for tandem duplications [10], Jain *et al.* demonstrated Lemma 7 and the fact that Φ and Ψ are bijections when $t = 2b - 1$. It is straightforward to extend the proof for $t \geqslant b$. Hence, we have the following lemma that is useful in evaluating the size of the code $\mathbb{C}_2(n, b, t)$.

Lemma 8: For all n, b, t,

$$|\mathbb{C}_2(n, b, t)| = 2^b \cdot |\mathcal{R}(n - b, t - b)|.$$

The size of the code $\mathcal{R}(n, t)$ can be calculated using the results from Section III and by applying Lemma 8 for $b = 1$ to get that for all n and $t \leqslant n$,

$$|\mathcal{R}(n, t)| = \frac{|\mathbb{C}_1(n + 1, 1, t + 1)|}{2}.$$

We can now conclude with the following corollary.

Corollary 9: For all n, b, t,

$$|\mathbb{C}_2(n, b, t)| = 2^b \cdot \frac{|\mathbb{C}_1(n - b + 1, 1, t - b + 1)|}{2}.$$

Finally, according to Corollary 3 and Corollary 9 we get the following result.

Corollary 10: There exists a two-head b-burst-deletion-correcting code when the heads are positioned $t = \lceil \log(n) \rceil + b$ locations apart with redundancy of approximately $\log(e)/4 \approx 0.36$ bits.

B. Correcting a Burst of Length at Most b

The goal of this section is to design a code correcting a burst of at most b deletions using two heads. We follow the same ideas presented thus far and use the following construction.

Construction 11: Let $\mathbb{C}_3(n, \leqslant b, t)$ be a code of length n which is the intersection of the codes $\mathbb{C}_2(n, \ell, t)$ for $1 \leqslant \ell \leqslant b$. That is,

$$\mathbb{C}_3(n, \leqslant b, t) = \cap_{\ell=1}^{b} \mathbb{C}_2(n, \ell, t) = \{\boldsymbol{c} \in \mathbb{F}_2^n \mid L(\boldsymbol{c}, \ell) \leqslant t, \text{ for all } \ell \leqslant b\}.$$

The correctness of this construction is verified in the next theorem.

Theorem 12: The code $\mathbb{C}_3(n, \leqslant b, t)$ can correct up to b consecutive deletions using two heads at distance t.

Proof: According to the output length of each head we can easily determine the size ℓ of the burst of deletions. Since $\mathbb{C}_3(n, \leqslant b, t) \subseteq \mathbb{C}_2(n, \ell, t)$, we simply apply the decoding algorithm presented in Theorem 5 for the code $\mathbb{C}_3(n, \ell, t)$ in order to decode the stored codeword. ■

We now turn to the evaluation of the code cardinality of $\mathbb{C}_3(n, \leqslant b, t)$. We will not be able to provide an exact approximation for the redundancy of the code $\mathbb{C}_3(n, \leqslant b, t)$ as we did before. However, we will find a value of t for which the redundancy of the code is at most a single bit. For

this purpose, we follow similar ideas to the ones presented by Schoeny *et al.* [23] when studying the redundancy of the so-called *universal RLL constraint*. This will be proved in the next theorem.

Theorem 13: For all n, b, t,

$$|\mathbb{C}_3(n, \leqslant b, t)| \geqslant 2^n \left(1 - n \cdot \left(\frac{1}{2}\right)^{t-b}\right).$$

In particular, for $t = \lceil \log(n) \rceil + b + 1$ the redundancy of the code $\mathbb{C}_3(n, \leqslant b, t)$ is at most a single bit.

Proof: According to the construction of the code $\mathbb{C}_3(n, \leqslant b, t) = \cap_{\ell=1}^{b} \mathbb{C}_2(n, \ell, t)$, we have that

$$|\mathbb{C}_3(n, \leqslant b, t)| = 2^n - |\overline{\mathbb{C}_3(n, \leqslant b, t)}| = 2^n - |\overline{\cap_{\ell=1}^{b} \mathbb{C}_2(n, \ell, t)}|$$
$$\geqslant 2^n - \sum_{\ell=1}^{b} |\overline{\mathbb{C}_2(n, \ell, t)}|.$$

For $1 \leqslant \ell \leqslant b$, a vector $\boldsymbol{c}$ belongs to $\overline{\mathbb{C}_2(n, \ell, t)}$ if and only if it consists of a subvector of period ℓ and length $t+1$. There are at most $n - t$ starting positions for such a subvector. Once we set the first ℓ bits in this subvector of length $t+1$, the rest of its bits are determined uniquely. The remaining $n - (t+1)$ bits in the vector can be chosen arbitrarily. Hence, according to the union bound, we get that the number of vectors in $\overline{\mathbb{C}_2(n, \ell, t)}$ is upper bounded by

$$|\overline{\mathbb{C}_2(n, \ell, t)}| \leqslant (n - t) \cdot 2^{\ell} \cdot 2^{n-(t+1)} \leqslant 2^n \cdot n \cdot \left(\frac{1}{2}\right)^{t-\ell+1}.$$

Hence,

$$\sum_{\ell=1}^{b} |\overline{\mathbb{C}_2(n, \ell, t)}| \leqslant \sum_{\ell=1}^{b} 2^n \cdot n \cdot \left(\frac{1}{2}\right)^{t-\ell+1}$$
$$= 2^n \cdot n \cdot \frac{2^b - 1}{2^t} \leqslant 2^n \cdot n \cdot \left(\frac{1}{2}\right)^{t-b}.$$

Therefore,

$$|\mathbb{C}_3(n, \leqslant b, t)| \geqslant 2^n - \sum_{\ell=1}^{b} |\overline{\mathbb{C}_2(n, \ell, t)}|$$
$$\geqslant 2^n - 2^n \cdot n \cdot \left(\frac{1}{2}\right)^{t-b} = 2^n \left(1 - n \cdot \left(\frac{1}{2}\right)^{t-b}\right).$$

In particular, for $t = \lceil \log(n) \rceil + b + 1$, we get that

$$|\mathbb{C}_3(n, \leqslant b, \lceil \log(n) \rceil + b + 1)| \geqslant 2^n \left(1 - n \cdot \left(\frac{1}{2}\right)^{\log(n)+b+1-b}\right)$$
$$= 2^{n-1},$$

and thus the redundancy of the last code is at most a single bit. ∎

V. Codes Correcting Multiple Deletions

In this section we move to the more challenging task of correcting multiple deletions and construct m-head d-deletion-correcting codes. For simplification, we first consider the case $d = 2$ and show that the code $\mathbb{C}_3(n, \leqslant 2, t_1)$, which can correct a burst of at most two deletions by using two heads, is a three-head double-deletion-correcting code, when the distance between every adjacent heads is at least $t = 2(t_1 - 1)$. We will then use this result as a building block for a more general claim on codes which can correct d deletions using $d + 1$ heads, that is, $(d+1)$-head d-deletion-correcting codes. Lastly, we show that the case of $d + 1$ heads is a special case of a more general result which claims that if the number of heads is $m \leqslant d + 1$ then it is possible to correct $m - 1$ deletions using the m heads. Hence, in order to correct d deletions, the stored codeword should belong to a code correcting $d - m + 1$ deletions. While we do not design new code constructions, a key point in the construction is finding the required minimum distance between two adjacent heads for its success.

A. Three-Head Double-Deletion-Correcting Codes

We start by presenting our result for the construction of three-head double-deletion-correcting codes.

Theorem 14: The code $\mathbb{C}_3(n, \leqslant 2, t_1)$ is a three-head double-deletion-correcting code when the distance between adjacent heads is at least $t = 2(t_1 - 1)$.

Proof: Let $\boldsymbol{c} = (c_1, \ldots, c_n) \in \mathbb{C}_3(n, \leqslant 2, t_1)$ be the stored codeword and $t = 2(t_1 - 1)$ be the distance between adjacent heads. Let us assume that the two deletions occurred in the first head are in positions i_1, i_2, where $i_1 < i_2$. Hence the deletions in the second head are in positions $i_1 + t, i_2 + t$ and in the third head they are in positions $i_1 + 2t, i_2 + 2t$. That is, the outputs from the three heads are:

Head 1 : $\boldsymbol{c}(\delta_{i_1}, \delta_{i_2})$
$= (c_1, \ldots, c_{i_1-1}, c_{i_1+1}, \ldots, c_{i_2-1}, c_{i_2+1}, \ldots, c_n)$,

Head 2 : $\boldsymbol{c}(\delta_{i_1+t}, \delta_{i_2+t})$
$= (c_1, \ldots, c_{i_1+t-1}, c_{i_1+t+1}, \ldots, c_{i_2+t-1}, c_{i_2+t+1}, \ldots, c_n)$,

Head 3 : $\boldsymbol{c}(\delta_{i_1+2t}, \delta_{i_2+2t})$
$= (c_1, \ldots, c_{i_1+2t-1}, c_{i_1+2t+1}, \ldots, c_{i_2+2t-1}, c_{i_2+2t+1}, \ldots, c_n)$.

We prove that it is possible to correct the two deletions by explicitly showing how to decode them. This will be done in three steps:

1) First, we use the first two heads to correct the first deletion in the first head.
2) Then, we use the second and third heads to correct the first deletion in the second head.
3) At this point, the first and second heads have only a single deletion and thus we proceed to correct this deletion as was done in Theorem 2.

We start with the first step and show how to correct the first deletion in the first head. To accomplish this task, we show that $\boldsymbol{c}(\delta_{i_1}, \delta_{i_2})[1, i_1 + t - 1] \neq \boldsymbol{c}(\delta_{i_1+t}, \delta_{i_2+t})[1, i_1 + t - 1]$. Assume the contrary, then we distinguish between the following two cases:

- **Case 1.1**: If $i_2 - i_1 \geqslant t_1 + 1$ then the two subvectors

$$\boldsymbol{c}(\delta_{i_1}, \delta_{i_2})[1, i_1 + t_1 - 1]$$
$$= (c_1, \ldots, c_{i_1-1}, c_{i_1+1}, c_{i_1+2}, \ldots, c_{i_1+t_1}),$$
$$\boldsymbol{c}(\delta_{i_1+t}, \delta_{i_2+t})[1, i_1 + t_1 - 1]$$
$$= (c_1, \ldots, c_{i_1-1}, c_{i_1}, c_{i_1+1}, \ldots, c_{i_1+t_1-1})$$

are identical, and thus the subvector $(c_{i_1}, c_{i_1+1}, \ldots, c_{i_1+t_1-1}, c_{i_1+t_1})$ forms a run of length t_1+1, in contradiction to the construction of the code $\mathbb{C}_3(n, \leqslant 2, t_1)$.

- **Case 1.2**: If $i_2 - i_1 \leqslant t_1$ then $i_1 + t = i_1 + 2t_1 - 2 \geqslant i_2 + t_1 - 2$. Then the first $i_2 + t_1 - 3$ bits in the first two heads, which are the subvectors

$$\begin{aligned}&\boldsymbol{c}(\delta_{i_1}, \delta_{i_2})[1, i_2 + t_1 - 3]\\ &\quad = (c_1, \ldots, c_{i_1-1}, c_{i_1+1}, \ldots, c_{i_2-1}, c_{i_2+1}, \ldots, c_{i_2+t_1-1}),\\ &\boldsymbol{c}(\delta_{i_1+t}, \delta_{i_2+t})[1, i_2 + t_1 - 3]\\ &\quad = (c_1, \ldots, c_{i_1-1}, c_{i_1}, \ldots, c_{i_2-2}, c_{i_2-1}, \ldots, c_{i_2+t_1-3}),\end{aligned}$$

  are identical, which implies that $(c_{i_2-1}, c_{i_2}, \ldots, c_{i_2+t_1-1})$ is a subvector of length t_1+1 with period 2, again in contradiction to the construction of the code $\mathbb{C}_3(n, \leqslant 2, t_1)$.

Let j_1 be the leftmost index that $\boldsymbol{c}(\delta_{i_1}, \delta_{i_2})$ and $\boldsymbol{c}(\delta_{i_1+t}, \delta_{i_2+t})$ differ. Similarly to the argument in the proof of Theorem 2, such an index exists and $i_1 \leqslant j_1 \leqslant i_1 + t - 1$. To correct the first deletion in the first head, we concatenate the first j_1 bits in $\boldsymbol{c}(\delta_{i_1+t}, \delta_{i_2+t})$ and the last $n-1-j_1$ bits in $\boldsymbol{c}(\delta_{i_1}, \delta_{i_2})$. Now, there are two cases to consider.

- **Case 2.1**: If $j_1 < i_2 - 1$ then $\boldsymbol{c}(\delta_{i_1}, \delta_{i_2})[j_1, n-2] = (c_{j_1+1}, \ldots, c_{i_2-1}, c_{i_2+1}, \ldots, c_n)$ and $\boldsymbol{c}(\delta_{i_1+t}, \delta_{i_2+t})[1, j_1] = (c_1, \ldots, c_{j_1})$. Thus, $\boldsymbol{c}(\delta_{i_1+t}, \delta_{i_2+t})[1, j_1] \circ \boldsymbol{c}(\delta_{i_1}, \delta_{i_2})[j_1, n-2] = \boldsymbol{c}(\delta_{i_2})$.
- **Case 2.2**: If $j_1 \geqslant i_2 - 1$ then $\boldsymbol{c}(\delta_{i_1}, \delta_{i_2})[j_1, n-2] = (c_{j_1+2}, \ldots, c_n)$ and $\boldsymbol{c}(\delta_{i_1+t}, \delta_{i_2+t})[1, j_1] = (c_1, \ldots, c_{j_1})$. Thus, $\boldsymbol{c}(\delta_{i_1+t}, \delta_{i_2+t})[1, j_1] \circ \boldsymbol{c}(\delta_{i_1}, \delta_{i_2})[j_1, n-2] = \boldsymbol{c}(\delta_{j_1+1})$. Furthermore, in this case, $(c_{i_2-1}, c_{i_2}, \ldots, c_{j_1}, c_{j_1+1})$ is a subvector of length $j_1 - i_2 + 3$ with period 2. Hence, $j_1 - i_2 + 3 \leqslant t_1$, which provides that $j_1 + 1 \leqslant i_2 + t_1 - 2$.

Therefore, in both cases we can write

$$\boldsymbol{c}(\delta_{i_1+t}, \delta_{i_2+t})[1, j_1] \circ \boldsymbol{c}(\delta_{i_1}, \delta_{i_2})[j_1, n-2] = \boldsymbol{c}(\delta_{i_2+k_1}),$$

where $0 \leqslant k_1 \leqslant t_1 - 2$.

The second step is not different than the first one. In fact, we have the same problem in the sense of having two heads at distance t which both had two deletions. Hence, we apply the same proof to show that

$$\boldsymbol{c}(\delta_{i_1+2t}, \delta_{i_2+2t})[1, i_1 + 2t - 1] \neq \boldsymbol{c}(\delta_{i_1+t}, \delta_{i_2+t})[1, i_1 + 2t - 1],$$

and then correct the first deletion in the second head to get $\boldsymbol{c}(\delta_{i_2+t+k_2})$ where $0 \leqslant k_2 \leqslant t_1 - 2$.

Lastly, in the third step we have two vectors $\boldsymbol{c}(\delta_{i_2+k_1})$ and $\boldsymbol{c}(\delta_{i_2+t+k_2})$. Note that $(i_2 + t + k_2) - (i_2 + k_1) \geqslant (i_2 + t) - (i_2 + t_1 - 2) = t_1$. That is, the distance between the two deletions in the two vectors is at least t_1. Therefore, following the proof in Theorem 2, we can reconstruct the codeword $\boldsymbol{c}$ from the two vectors $\boldsymbol{c}(\delta_{i_2+k_1})$ and $\boldsymbol{c}(\delta_{i_2+t+k_2})$, thereby correcting two random deletions by using three heads of distance t between adjacent heads. ∎

The next example demonstrates the decoding procedure presented in Theorem 14.

Example 6: Let $n = 14, t_1 = 3, t = 4$ and

$$\boldsymbol{c} = (0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1)$$

be a stored codeword in $\mathbb{C}(14, \leqslant 2, 3)$. Assume that the outputs from three heads are:

$$\begin{aligned}&\textbf{Head 1}: \boldsymbol{c}(\delta_3, \delta_5) = (0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1)\\ &\textbf{Head 2}: \boldsymbol{c}(\delta_7, \delta_9) = (0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1)\\ &\textbf{Head 3}: \boldsymbol{c}(\delta_{11}, \delta_{13}) = (0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1).\end{aligned}$$

By comparing the outputs from first two heads, we see that $j_1 = 5$ is the leftmost index that $\boldsymbol{c}(\delta_3, \delta_5)$ and $\boldsymbol{c}(\delta_7, \delta_9)$ differ. Hence, we can obtain the vector

$$\begin{aligned}\boldsymbol{c}(\delta_6) &= \boldsymbol{c}(\delta_7, \delta_9)[1, 5] \circ \boldsymbol{c}(\delta_3, \delta_5)[5, 12]\\ &= (0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1).\end{aligned}$$

Similarly, we find the leftmost index that $\boldsymbol{c}(\delta_7, \delta_9)$ and $\boldsymbol{c}(\delta_{11}, \delta_{13})$ differ which is $j_2 = 7$ and obtain the vector

$$\begin{aligned}\boldsymbol{c}(\delta_9) &= \boldsymbol{c}(\delta_{11}, \delta_{13})[1, 7] \circ \boldsymbol{c}(\delta_7, \delta_9)[7, 12]\\ &= (0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1).\end{aligned}$$

Note that when decoding the first deletion in the first head the condition $j_1 < i_2 - 1$ does not hold so we are in Case 2.2 and thus the remaining deletion is in location 6 instead of 5. However, this condition does hold when decoding the first deletion in the second head and hence the deletion is in position 9 (Case 2.1). Lastly, we can recover the original codeword by finding $j_3 = 7$ as the leftmost index that $\boldsymbol{c}(\delta_6)$ and $\boldsymbol{c}(\delta_9)$ differ and recover the stored codeword $\boldsymbol{c}$ to be

$$\begin{aligned}\boldsymbol{c} &= \boldsymbol{c}(\delta_9)[1, 7] \circ \boldsymbol{c}(\delta_6)[7, 13]\\ &= (0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1).\end{aligned}$$

□

Although Theorem 13 provides a good approximation on the cardinality of the code $\mathbb{C}_3(n, \leqslant b, t_1)$ for all b, the following lemma gives a better result on the special case with $b = 2$.

Lemma 15: For all n, t,

$$|\mathbb{C}_3(n, \leqslant 2, t_1)| = |\mathbb{C}_2(n, 2, t_1)| = 2 \cdot |\mathbb{C}_1(n-1, 1, t_1 - 1)|.$$

Proof: We observe that any vector $(x_1, \ldots, x_\ell)$ which has period 1, that is $x_1 = x_2 = \cdots = x_\ell$, has also period 2. Hence, $\mathbb{C}_2(n, 2, t_1) \subseteq \mathbb{C}_2(n, 1, t_1)$. Therefore,

$$\mathbb{C}_3(n, \leqslant 2, t_1) = \mathbb{C}_2(n, 2, t_1) \cap \mathbb{C}_2(n, 1, t_1) = \mathbb{C}_2(n, 2, t_1).$$

Moreover, from Corollary 9, we obtain

$$|\mathbb{C}_2(n, 2, t_1)| = 2 \cdot |\mathbb{C}_1(n-1, 1, t_1 - 1)|.$$

Lastly, from the equations above, Lemma 15 is proven. ∎

Based on the results in Lemma 15 and Corollary 3 we conclude with the following corollary.

Corollary 16: There exists a three-head double-deletion-correcting code with redundancy of approximately $\log(e)/4 \approx 0.36$ *bits when the distance between adjacent heads is at least* $t = 2(\lceil \log(n) \rceil + 1)$.

B. Multiple-Head b-Deletion-Correcting Codes

The idea in the proof of Theorem 14 is to use every two pairs of subvectors in order to correct the first deletion in the first subvector in each pair. It turns that this basic procedure can be generalized to d random deletions which we need to construct the m-head d-deletion-correcting codes. We start with the case of $m = d + 1$ which allows to correct all the deletions and then discuss the arbitrary case of $m \leqslant d$.

First we show how to generalize the procedure of using two adjacent heads to correct the first deletion in the first head for the case of arbitrary number of deletions d.

Lemma 17: Let d and t_1 be two positive integers such that $t_1 > d$ and let $t = dt_1 - d(d+1)/2 + 1$. For $h = 1, 2$ let $\Delta_h = \{\delta_{i_{h,1}}, \ldots, \delta_{i_{h,d}}\}$ be two sets of deletion locations, such that for $\ell \in [d]$,

$$i_{2,\ell} - i_{1,\ell} \geqslant t = dt_1 - d(d+1)/2 + 1. \tag{3}$$

Assume $\boldsymbol{c} \in \mathbb{C}_3(n, \leqslant d, t_1)$ and the two vectors $\boldsymbol{c}(\Delta_1)$ and $\boldsymbol{c}(\Delta_2)$ are given. Then, it is possible to correct the first deletion in the first vector $\boldsymbol{c}(\Delta_1)$ and obtain the vector $\boldsymbol{c}(\Delta_1')$, where $\Delta_1' = \{\delta_{i'_{1,2}}, \ldots, \delta_{i'_{1,d}}\}$ and for $2 \leqslant \ell \leqslant d$,

$$i_{1,\ell} \leqslant i'_{1,\ell} \leqslant i_{1,\ell} + (d-1)t_1 - d(d-1)/2 + 1. \tag{4}$$

Proof: We will first show that

$$\boldsymbol{c}(\Delta_1)[1, i_{1,1} + t - 1] \neq \boldsymbol{c}(\Delta_2)[1, i_{1,1} + t - 1].$$

Assume the contrary that $\boldsymbol{c}(\Delta_1)[1, i_{1,1}+t-1] = \boldsymbol{c}(\Delta_2)[1, i_{1,1}+t-1]$, and we prove the following proposition.

Proposition 18: For all $1 \leqslant k \leqslant d-1$,

$$\begin{aligned} i_{1,k+1} \leqslant i_{1,k} + t_1 - k + 1 &\leqslant i_{1,1} + kt_1 - \sum_{i=1}^{k-1} i \\ &= i_{1,1} + kt_1 - k(k-1)/2. \end{aligned}$$

Proof: We prove this property by induction. For the base we prove that it holds for $k = 1$. Assume the contrary that the claim does not hold, that is, $i_{1,2} \geqslant i_{1,1} + t_1 + 1$. Then, the following two subvectors

$$\begin{aligned} &\boldsymbol{c}(\Delta_1)[1, i_{1,1} + t_1 - 1] \\ &\qquad = (c_1, \ldots, c_{i_{1,1}-1}, c_{i_{1,1}+1}, c_{i_{1,1}+2}, \ldots, c_{i_{1,1}+t_1}), \\ &\boldsymbol{c}(\Delta_2)[1, i_{1,1} + t_1 - 1] \\ &\qquad = (c_1, \ldots, c_{i_{1,1}-1}, c_{i_{1,1}}, c_{i_{1,1}+1}, \ldots, c_{i_{1,1}+t_1-1}) \end{aligned}$$

are identical since $i_{1,1} + t_1 - 1 \leqslant i_{1,1} + t - 1$. That is, the subvector $(c_{i_{1,1}}, c_{i_{1,1}+1}, \ldots, c_{i_{1,1}+t_1-1}, c_{i_{1,1}+t_1})$ is a run of length $t_1 + 1$, which is a contradiction since $\boldsymbol{c} \in \mathbb{C}_3(n, \leqslant d, t_1)$.

Next we assume that this property holds for all k such that $1 \leqslant k < k_0 \leqslant d-1$ and we prove its correctness for k_0. Notice that according to the induction assumption for $k = k_0 - 1$ we have that

$$i_{1,k_0} \leqslant i_{1,1} + (k_0 - 1)t_1 - \sum_{i=1}^{k_0-2} i,$$

and therefore

$$i_{1,k_0} + t_1 - k_0 + 1 \leqslant i_{1,1} + k_0 t_1 - \sum_{i=1}^{k_0-1} i,$$

which implies that

$$i_{1,k_0} + t_1 - k_0 + 1 \leqslant i_{1,1} + t - 1.$$

Assume the contrary that $i_{1,k_0+1} > i_{1,k_0} + t_1 - k_0 + 1$. Then the following two subvectors

$$\begin{aligned} &\boldsymbol{c}(\Delta_1)[i_{1,k_0} - k_0, i_{1,k_0} + t_1 - 2k_0 + 1] \\ &\qquad = (c_{i_{1,k_0}-1}, c_{i_{1,k_0}+1}, \ldots, c_{i_{1,k_0}+t_1-k_0+1}) \\ &\boldsymbol{c}(\Delta_2)[i_{1,k_0} - k_0, i_{1,k_0} + t_1 - 2k_0 + 1] \\ &\qquad = (c_{i_{1,k_0}-k_0}, c_{i_{1,k_0}-k_0+1}, \ldots, c_{i_{1,k_0}+t_1-2k_0+1}) \end{aligned}$$

are identical. That is, the subvector $(c_{i_{1,k_0}-k_0+1}, \ldots, c_{i_{1,k_0}+t_1-k_0+1})$ is a subvector with period k_0 of length $t_1 + 1$, which leads again to a contradiction. Therefore, we deduce that $i_{1,k_0+1} \leqslant i_{1,k_0} + t_1 - k_0 + 1$, and the claim holds also for $k = k_0 + 1$. This completes the proof. ∎

According to Proposition 18, we obtain that for $k = d-1$,

$$i_{1,d} \leqslant i_{1,1} + (d-1)t_1 - (d-1)(d-2)/2.$$

Therefore,

$$\begin{aligned} i_{1,d} + t_1 - 2d + 1 &\leqslant i_{1,1} + v(d-1)t_1 - (d-1)(d-2)/2 + t_1 - 2d + 1 \\ &= i_{1,1} + t - 1. \end{aligned}$$

Hence, the following two vectors

$$\begin{aligned} &\boldsymbol{c}(\Delta_1)[i_{1,d} - d, i_{1,d} + t_1 - 2d + 1] \\ &\quad = (c_{i_{1,d}-1}, c_{i_{1,d}+1}, \ldots, c_{i_{1,d}+t_1-d+1}), \\ &\boldsymbol{c}(\Delta_2)[i_{1,d} - d, i_{1,d} + t_1 - 2d + 1] \\ &\quad = (c_{i_{1,d}-d}, c_{i_{1,d}-d+1}, \ldots, c_{i_{1,d}+t_1-2d+1}) \end{aligned}$$

are also identical. Again we get a contradiction since the subvector $(c_{i_{1,d}-d+1}, \ldots, c_{i_{1,d}+t_1-d+1})$ with period d is of length $t_1 + 1$. Therefore,

$$\boldsymbol{c}(\Delta_1)[1, i_{1,1} + t - 1] \neq \boldsymbol{c}(\Delta_2)[1, i_{1,1} + t - 1].$$

Let j_1 be the leftmost index where the last two vectors $\boldsymbol{c}(\Delta_1)$ and $\boldsymbol{c}(\Delta_2)$ differ. Similarly to the argument in the proof of Theorem 2, we know that such an index exists and

$$i_{1,1} \leqslant j_1 \leqslant i_{1,1} + t - 1. \tag{5}$$

To correct the first deletion in $\boldsymbol{c}(\Delta_1)$, we concatenate the first j_1 bits from $\boldsymbol{c}(\Delta_2)$ with the last $n-d-j_1+1$ bits from $\boldsymbol{c}(\Delta_1)$, that is, we form the vector

$$\boldsymbol{c}(\Delta_1') = \boldsymbol{c}(\Delta_2)[1, j_1] \circ \boldsymbol{c}(\Delta_1)[j_1, n-d],$$

which has $d-1$ deletions that we denote by the set

$$\Delta_1' = \{\delta_{i'_{1,2}}, \ldots, \delta_{i'_{1,d}}\}.$$

Note that $j_1 > i_{1,1} - 1$ and $i_{1,1} - 1 \leqslant i_{1,2} - 2 \leqslant \cdots \leqslant i_d - d$ since $i_{1,1} < i_{1,2} < \cdots < i_{1,d}$. Hence, there exists a unique index $r \in [d-1]$ such that $i_{1,r} - r < j_1 \leqslant i_{1,r+1} - (r+1)$ or $j_1 > i_{1,d} - d$. We now consider the following two cases.

- **Case 1**: There exists an index $r \in [d-1]$ such that $i_{1,r} - r < j_1 \leqslant i_{1,r+1} - (r+1)$. In this case, $\boldsymbol{c}(\Delta_1)[j_1] = c_{j_1+r}$

and $i_{1,1} < \cdots < i_{1,r} < j_1 + r < i_{1,r+1} < \cdots < i_{1,d}$. Hence,

$$\boldsymbol{c}(\Delta_1)[j_1, n-d] = \boldsymbol{c}[j_1 + r, n](\{\delta_{i_{1,r+1}}, \ldots, \delta_{i_{1,d}}\}).$$

That is, the locations of the last $d-r$ deletions did not change. Furthermore, according to (5) we get

$$\boldsymbol{c}(\Delta_2)[1, j_1] = \boldsymbol{c}[1, j_1] = (c_1, \ldots, c_{j_1}).$$

Therefore,

$$\boldsymbol{c}(\Delta_2)[1, j_1] \circ \boldsymbol{c}(\Delta_1)[j_1, n-d] = \boldsymbol{c}(\Delta_1')$$

where

$$\begin{aligned}\Delta_1' &= \{\delta_{i_{1,2}'}, \ldots, \delta_{i_{1,d}'}\} \\ &= \{\delta_{j_1+1}, \ldots, \delta_{j_1+r-1}, \delta_{i_{1,r+1}}, \ldots, \delta_{1,i_{1,d}}\}.\end{aligned}$$

Now, we need to prove that $i_{1,\ell} \leqslant i_{1,\ell}' \leqslant i_{1,\ell} + (d-1)t_1 - d(d-1)/2 + 1$ for $2 \leqslant \ell \leqslant d$. For $2 \leqslant \ell \leqslant r$, $i_{1,\ell} \leqslant j_1 + \ell - 1 = i_{1,\ell}'$, since $j_1 > i_r - r \geqslant \cdots \geqslant i_{1,\ell} - \ell$. Furthermore, since $j_1 + r < i_{1,r+1}$, we have that $i_{1,\ell}' = j_1 + \ell - 1 \leqslant j_1 + r - 1 \leqslant i_{1,r+1}$ and from Proposition 17,

$$\begin{aligned}i_{1,\ell}' &\leqslant i_{1,r+1} \leqslant i_{1,r} + t_1 - (r-1) \\ &\leqslant i_{1,r-1} + t_1 - (r-1) + t_1 - (r-2) \\ &\leqslant i_{1,\ell} + \sum_{u=\ell-1}^{r-1} (t_1 - u) \\ &= i_{1,\ell} + \sum_{u=1}^{d-1} (t_1 - u) \leqslant i_{1,\ell} + (d-1)t_1 - d(d-1)/2 + 1.\end{aligned}$$

Lastly, for $r+1 \leqslant \ell \leqslant d$, $i_{1,\ell}' = i_{1,\ell}$, so we conclude that the property in (4) holds in this case.

- **Case 2**: $j_1 > i_{1,d} - d$. In this case, $\boldsymbol{c}(\Delta_1)[j_1, n-d] = \boldsymbol{c}[j_1 + d, n]$ and $\boldsymbol{c}(\Delta_2)[1, j_1] = \boldsymbol{c}[1, j_1]$. Hence,

$$\boldsymbol{c}(\Delta_2)[1, j_1] \circ \boldsymbol{c}(\Delta_1)[j_1, n-d] = \boldsymbol{c}(\Delta_1')$$

where

$$\Delta_1' = \{\delta_{i_{1,2}'}, \ldots, \delta_{i_{1,d}'}\} = \{\delta_{j_1+1}, \ldots, \delta_{j_1+d-1}\}.$$

Similarly to the proof of Proposition 18, we consider the two identical vectors

$$\begin{aligned}\boldsymbol{c}(\Delta_1)[i_{1,d} - d, j_1 - 1] &= (c_{i_{1,d}-1}, c_{i_{1,d}+1}, \ldots, c_{j_1+d-1}), \\ \boldsymbol{c}(\Delta_2)[i_{1,d} - d, j_1 - 1] &= (c_{i_{1,d}-d}, c_{i_{1,d}-d+1}, \ldots, c_{j_1-1})\end{aligned}$$

which imply that the subvector $(c_{i_{1,d}-d+1}, c_{i_{1,d}-d+2}, \ldots, c_{j_1+d-1})$ of length $j_1 - i_{1,d} + 2d - 1$ has period d. Therefore $j_1 - i_{1,d} + 2d - 1 \leqslant t_1$ or $j_1 \leqslant i_{1,d} + t_1 - d + 1$. For all $2 \leqslant \ell \leqslant d-1$, as in Case 1, we have that

$$i_{1,d} \leqslant i_{1,\ell} + \sum_{u=\ell-1}^{d-2} (t_1 - u),$$

and together we conclude that

$$\begin{aligned}i_{1,\ell}' &= j_1 + \ell - 1 \leqslant i_{1,d} + t_1 - d + 1 + \ell - 1 \\ &\leqslant i_{1,\ell} + \sum_{u=\ell-1}^{d-2} (t_1 - u) + t_1 - d + \ell \\ &= i_{1,\ell} + (d - \ell + 1)t_1 - \left(\sum_{u=\ell-1}^{d-2} u\right) - (d-1) + (\ell - 1) \\ &= i_{1,\ell} + (d - \ell + 1)t_1 - \left(\sum_{u=\ell}^{d-1} u\right) \\ &\leqslant i_{1,\ell} + (d-1)t_1 - d(d-1)/2 + 1.\end{aligned}$$

To summarize, we get that in both cases, $i_{1,\ell} \leqslant i_{1,\ell}' \leqslant i_{1,\ell} + (d-1)t_1 - d(d-1)/2 + 1$ for all $2 \leqslant \ell \leqslant d$, which verifies the correctness of Lemma 17. ∎

For the rest of this section we assume that d and t_1 are two positive integers such that $t_1 > d$ and the value of $T(d)$ is given by

$$\begin{aligned}T(d) &= t_1 + \sum_{k=1}^{d} \left((k-1)t_1 - k(k-1)/2 + 1\right) \\ &= t_1 \left(\frac{d(d-1)}{2} + 1\right) + \frac{7d - d^3}{6}.\end{aligned}$$

Before constructing $(d+1)$-head d-deletion-correcting codes, we provide the following claim.

Claim 19: Let $\Delta_h = \{\delta_{i_{h,1}}, \ldots, \delta_{i_{h,d}}\}$ for $h \in [d+1]$ be such that $i_{h,1} < i_{h,2} < \cdots < i_{h,d}$ where for given $\ell, h \in [d]$, it holds that $i_{h+1,\ell} - i_{h,\ell} \geqslant T(d)$. Assume the $d+1$ vectors $\boldsymbol{c}(\Delta_h)$ for $h \in [d+1]$ are given, where $\boldsymbol{c} \in \mathbb{C}_3(n, \leqslant d, t_1)$. Then, it is possible to decode the codeword $\boldsymbol{c}$.

Proof: We prove this claim by induction on d. For the base when $d = 1$, we need to prove that using two vectors, $\boldsymbol{c}(\delta_{i_{1,1}})$ and $\boldsymbol{c}(\delta_{i_{2,1}})$, the codeword $\boldsymbol{c}$ can be successfully decoded. This is proven in Theorem 2.

Next, we assume that Claim 19 holds for all $1 \leqslant d \leqslant d_0 - 1$, and we show that it holds also for $d = d_0$. For $h \in [d_0 + 1]$, let $\Delta_h = \{\delta_{i_{h,1}}, \ldots, \delta_{i_{h,d_0}}\}$ be $d_0 + 1$ sets which satisfy the properties in the claim. For $h \in [d_0]$, let us consider the two vectors $\boldsymbol{c}(\Delta_h)$ and $\boldsymbol{c}(\Delta_{h+1})$. According to Lemma 17, it is possible to obtain the vector $\boldsymbol{c}(\Delta_h')$ where $\Delta_h' = \{\delta_{i_{h,2}'}, \ldots, \delta_{i_{h,d_0}'}\}$ and for $2 \leqslant \ell \leqslant d_0$, $i_{h,\ell} \leqslant i_{h,\ell}' \leqslant i_{h,\ell} + (d_0 - 1)t_1 - d_0(d_0 - 1)/2 + 1$. Furthermore, since $i_{h+1,\ell} - i_{h,\ell} \geqslant T(d_0)$, we get that for all $h, \ell \in [d_0 - 1]$,

$$\begin{aligned}i_{h+1,\ell}' - i_{h,\ell}' &\geqslant i_{h+1,\ell} - \left(i_{h,\ell} + (d_0 - 1)t_1 - d_0(d_0 - 1)/2 + 1\right) \\ &\geqslant T(d_0) - \left((d_0 - 1)t_1 - d_0(d_0 - 1)/2 + 1\right) \\ &= T(d_0 - 1).\end{aligned}$$

Thus, according to the induction assumption for the d_0 sets Δ_h', $h \in [d_0]$, we can successfully decode the codeword $\boldsymbol{c}$, which proves the statement in the claim. ∎

Now we are ready to present our construction of $(d+1)$-head d-deletion-correcting codes.

Theorem 20: The code $\mathbb{C}_3(n, \leqslant d, t_1)$ is a $(d+1)$-head d-deletion-correcting code where the distance between adjacent heads is $t \geqslant T(d)$.

Proof: Let $\boldsymbol{c} \in \mathbb{C}_3(n, \leqslant d, t_1)$ be a stored codeword. For $h \in [d+1]$, let Δ_h be a set of d deletions occurred in the h-th head. Since the distance between adjacent heads is $t \geqslant T(d)$, all $d+1$ sets Δ_h satisfy the condition in Claim 19, and therefore we can successfully decode the codeword $\boldsymbol{c}$. ■

We note that the code that we use to correct up to d random deletions using $d+1$ heads is the same code used to correct a burst of at most d deletions using two heads, however here we require the distance between adjacent heads to be larger. The following corollary summarizes this discussion and analyzes the redundancy of the construction.

Corollary 21: There exists a $(d+1)$-head d-deletion-correcting code with at most a single bit of redundancy when the distance between adjacent heads is at least $\left(\binom{d}{2}+1\right) \cdot \lceil \log n \rceil + \frac{d^3+5d+3}{3}$.

We now turn to the case where the number of heads is less than $d+1$. In this case we can still follow the same logic of the proof in Theorem 20. However, now it is possible to correct only the first $m-1$ deletions. Thus if the stored codeword belongs to a $(d-m+1)$-deletion-correcting code, then it will be possible to correct all d deletions. This result is stated in the next theorem.

Theorem 22: Let $\mathbb{C}$ be a $(d-m+1)$-deletion-correcting code, where $m \leqslant d$. Then, the code $\mathbb{C} \cap \mathbb{C}_3(n, \leqslant d, t_1)$ is an m-head d-deletion-correcting code where the distance between adjacent heads is $t \geqslant T(d)$. In particular under this setup, if $t_1 = \lceil \log(n) \rceil + d + 1$ then:

1) *There exists a $(d+1)$-head d-deletion-correcting code with at most a single bit of redundancy*
2) *There exists a d-head d-deletion-correcting code with redundancy at most $\lceil \log(n+1) \rceil + 1$.*

The second part of Theorem 22 is proved by simply using the Varshamov-Tenengolts codes for single-deletion correction [25], which form a partition of the space. In order to correct more than a single deletion, one can use multiple-deletion correcting codes in [1], however there is no explicit expression for their redundancy.

VI. Codes Correcting Multiple Bursts of Sticky Insertions

In this section, we consider the case that there are only sticky insertions and construct codes correcting multiple sticky insertions using multiple heads. Recall that a sticky insertion occurs when the domain is not shifted and the same bit is read again by the head. Furthermore, if the domain does not move on several consecutive shift operations, then the same bit might be read multiple times in a row by the head and thus a burst of sticky insertions occurs.

For a length-n word $\boldsymbol{u} \in \mathbb{F}_2^n$ and $i \in [n]$, we denote by $\boldsymbol{u}(\gamma_{[i,b]})$ the vector obtained by $\boldsymbol{u}$ after repeating its ith bit b times, that is,

$$\boldsymbol{u}(\gamma_{[i,b]}) = (u_1, \ldots, u_{i-1}, \underbrace{u_i, \ldots, u_i}_{b+1 \text{ times}}, u_{i+1}, \ldots, u_n).$$

In case $b = 1$, we simply use the notation $\boldsymbol{u}(\gamma_i)$ instead of $\boldsymbol{u}(\gamma_{[i,1]})$. For a set $\Gamma \subseteq \{\gamma_{[i,b_i]} : i \in [n], b_i \geqslant 1\}$, we denote by $\boldsymbol{u}(\Gamma)$ the vector obtained from $\boldsymbol{u}$ after repeating its ith bit b_i times for all i, b_i such that $\gamma_{[i,b_i]} \in \Gamma$.

Example 7: Let $\boldsymbol{u} = (0,0,1,1,0,1,1) \in \mathbb{F}_2^7$, then $\boldsymbol{u}(\gamma_{[4,3]}) = (0,0,1,1,1,1,1,0,1,1)$. For $\Gamma = \{\gamma_{[1,1]}, \gamma_{[4,2]}\}$ then $\boldsymbol{u}(\Gamma) = (0,0,0,1,1,1,1,0,1,1)$. □

Recall again that in a racetrack memory, we use multiple heads in fixed positions to read the information so each bit is read multiple times. Therefore, if a sticky insertion occurs at the i-th position in the first head then in the second head it appears in the $(i+t)$-th position. That is, if $\boldsymbol{u}$ is the stored codeword and the output from the first head is $\boldsymbol{u}(\gamma_{[i,b]})$, then the output from the second head is $\boldsymbol{u}(\gamma_{[i+t,b]})$.

Although codes correcting a single sticky insertion are well-studied and asymptotically optimal codes exist, the redundancy of such codes is at least $\log(n) - 1$ bits [6], [18]. The main result in this section shows that using multiple heads, it is possible to correct multiple bursts of sticky insertions with at most a single bit of redundancy.

We observe that correcting a sticky insertion is an easier task than correcting a deletion. In fact, the code $\mathbb{C}_1(n,1,t)$, which is a two-head single-deletion-correcting code when the distance between the heads is at least t, is capable of correcting a single sticky insertion under the same setup. However, the next theorem shows that this code is actually capable of correcting a burst of sticky insertions of length at most $t-1$.

Theorem 23: The code $\mathbb{C}_1(n,1,t)$ is a two-head b-burst-sticky-insertion-correcting code for $b \leqslant t-1$ using two heads of distance t.

Proof: Let $\boldsymbol{c} = (c_1, \ldots, c_n) \in \mathbb{C}_1(n,1,t)$ be a stored codeword and assume that the output from the first, second head is $\boldsymbol{c}(\gamma_{[i,b]})$ and $\boldsymbol{c}(\gamma_{[i+t,b]})$, where $b \leqslant t-1$, respectively. Hence the outputs from the two heads are given by:

$$\textbf{Head 1}: \boldsymbol{c}(\gamma_{[i,b]}) = (c_1, \ldots, c_{i-1}, \underbrace{c_i, c_i, \ldots, c_i}_{b+1 \text{ times}}, c_{i+1}, \ldots, c_{i+t-b-1}, c_{i+t-b}, c_{i+t-b+1}, \ldots, c_n),$$

$$\textbf{Head 2}: \boldsymbol{c}(\gamma_{[i+t,b]}) = (c_1, \ldots, c_{i-1}, c_i, \ldots, c_{i+b}, c_{i+b+1}, \ldots, c_{i+t-1}, \underbrace{c_{i+t}, \ldots, c_{i+t}}_{b+1 \text{ times}}, c_{i+t+1}, \ldots, c_n).$$

As in previous proofs, we claim that $\boldsymbol{c}(\gamma_{[i,b]})[1, i+t-1] \neq \boldsymbol{c}(\gamma_{[i+t,b]})[1, i+t-1]$. Assume the contrary that equality holds, then we will get that

$$c_i = c_{i+1} = \cdots = c_{i+b+1} = \cdots = c_{i+t},$$

which implies a run of length $t+1$ in $\boldsymbol{c}$, a contradiction. Let us choose j to be the leftmost index such that $\boldsymbol{c}(\gamma_{[i,b]})$ and $\boldsymbol{c}(\gamma_{[i+t,b]})$ differ, and note that as before $i+1 \leqslant j \leqslant i+t-1$. Since we know that the error is a burst of sticky insertions of length b and it first occurs in the head first, we can recover the stored codeword by simply deleting the b consecutive bits $\boldsymbol{c}(\gamma_{[i,b]})[j, j+b-1]$ in $\boldsymbol{c}(\gamma_{[i,b]})$. ■

The next corollary is a direct result of Theorem 23 and the results stated in Section III

Corollary 24: There exists a two-head b-burst-sticky-insertion-correcting code for $b \leqslant t-1$, where $t = \lceil \log(n) \rceil + 1$, when the distance between the heads is t. The redundancy of the code is approximately 0.36 bits.

Next we extend this construction to correct multiple burst of sticky insertions using multiple heads. We follow the same logic as in Section V. Every two adjacent heads will allow to correct the first burst of sticky insertions in the first head and if there are m heads it is possible to correct $m-1$ bursts of sticky insertions. Similarly to Section V, we also generalize the last construction to codes correcting multiple bursts of sticky insertions using multiple heads. This property is formally stated as follows. We omit the proof since it follows the same one of Theorem 23.

Theorem 25: For every $\boldsymbol{c} \in \mathbb{C}_1(n,1,t)$, and two adjacent heads of distance at least t, if there occurred some d bursts of sticky insertions, each of length at most $t-1$, then it is possible to correct the first burst of sticky insertions in the first head.

Finally, we can conclude with the following corollary.

Corollary 26: The code $\mathbb{C}_1(n,1,t)$ can correct d bursts of sticky insertions each of length at most $t-1$ using $d+1$ heads while the distance between adjacent heads is at least t. Specifically, for $t = \lceil \log(n) \rceil + 1$, the redundancy of the code is approximately 0.36 bits.

Remark 1: In the classical model of insertions and deletions it is known that correcting deletions is equivalent to correcting insertions. However, correcting sticky insertions is no longer equivalent to correcting deletions and indeed this is a significantly easier problem [6], [18]. This is also the case in our study and indeed correcting sticky insertions is also shown to be an easier problem than correcting deletions. We observe that $\mathbb{C}_3(n, \leqslant d, t_1) \subset \mathbb{C}_1(n,1,t_1)$, where the former code is used to correct d deletions, while the latter can correct d bursts of sticky insertions, and both codes use $d+1$ heads. Furthermore, recall that our model is a special model of Levenshtein's reconstruction problem. In the reconstruction problem, Levenshtein [12], [13] and recently Sala *et al.* [20] showed that the solution of the reconstruction problem for insertions is not equivalent to the one for deletions. Lastly, in our model, we also observe that it is much more difficult to correct the two types of errors, that is, deletions and sticky insertions. In next section, we investigate this problem in more detail.

VII. Codes Correcting Combination of Deletions and Sticky Insertions

In this section, we tackle the more difficult problem of constructing m-head d-position-error-correcting codes in which position errors can be deletions or sticky insertions. If both deletions and sticky insertions occur, we can determine the difference between the number of deletions and number of sticky insertions. Thus, if there exists only a single position error, we can determine whether that position error is a deletion or sticky insertion. Moreover, we already proved that the code $\mathbb{C}_1(n,1,t)$ is a two-head single-deletion-correcting code if the distance between two heads is at least t, and under this setup from Theorem 23, the code $\mathbb{C}_1(n,1,t)$ is also a two-head single-sticky-insertion-correcting code since it can correct a burst of sticky insertions. Therefore, we obtain the following result.

Theorem 27: The code $\mathbb{C}_1(n,1,t)$ is a two-head single-position-error-correcting code if the distance between the two heads is at least t.

Our next step is to construct a three-head two-position-error-correcting code.

Theorem 28: The code $\mathbb{C}_3(n, \leqslant 2, t_1)$ is a three-head two-position-error-correcting code if the distance between adjacent heads is at least $t = 3t_1 - 2$.

Proof: Let d_1 be the number of deletions and d_2 be the number of sticky insertions. Since there are at most 2 position errors which can be deletions or sticky insertions, we obtain $(d_1, d_2) \in \{(0,0),(0,1),(1,0),(0,2),(2,0),(1,1)\}$. Recall that we always know what the different between d_1 and d_2 is. Since most of the cases have already been proved in previous sections, it is enough to consider the case that $d_1 - d_2 = 0$, that is, (d_1, d_2) can be $(0,0)$ or $(1,1)$.

Let $\boldsymbol{c} = (c_1, \ldots, c_n) \in \mathbb{C}_3(n, \leqslant 2, t)$ be the stored codeword, and let $\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_3 \in \{0,1\}^n$ be the output from the first, second, third head, respectively. In case the i_1-th bit is deleted and the i_2-th bit is repeated at the first head we get that $\boldsymbol{c}_1 = \boldsymbol{c}(\delta_{i_1}, \gamma_{i_2})$, $\boldsymbol{c}_2 = \boldsymbol{c}(\delta_{i_1+t}, \gamma_{i_2+t})$, $\boldsymbol{c}_3 = \boldsymbol{c}(\delta_{i_1+2t}, \gamma_{i_2+2t})$, in which $\boldsymbol{c}(\delta_i, \gamma_j)$ is a vector obtained from $\boldsymbol{c}$ by deleting c_i and repeating c_j. Note that we did not assume here that $i_1 < i_2$.

We prove the theorem by explicitly showing how to decode the stored codeword $\boldsymbol{c}$. This will be done in the following three steps:

Step 1: Determining whether $\boldsymbol{c}_1 = \boldsymbol{c}$.

Step 2: Determining whether the first error in the first head is a deletion or sticky insertion.

Step 3: Correcting two position errors to recover the stored codeword.

We start with the first step.

Step 1: In the next claim we present a necessary and sufficient condition to determine whether $\boldsymbol{c}_1 = \boldsymbol{c}$.

Claim 29: $\boldsymbol{c} = \boldsymbol{c}_1$ if and only if one of the following two conditions holds:

1) *$\boldsymbol{c}_1 = \boldsymbol{c}_2$, or*
2) *$\boldsymbol{c}_1 \neq \boldsymbol{c}_2$ and $\boldsymbol{c}_1[j_1] = \boldsymbol{c}_3[j_1] \neq \boldsymbol{c}_2[j_1]$, where j_1 is the leftmost index that $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ differ.*

Proof: We know that $\boldsymbol{c} = \boldsymbol{c}_1$ if and only if there is no error or there are two position errors in the first head which are a deletion in the i_1-th bit and a sticky insertion in the i_2-th bit but c_{i_1} and c_{i_2} are in the same run in $\boldsymbol{c}$.

Let us assume that $\boldsymbol{c} = \boldsymbol{c}_1$, then one of the two following cases happens.

(i) **Case 1**: There is no error, that is, $\boldsymbol{c}_1 = \boldsymbol{c}_2 = \boldsymbol{c}$.

(ii) **Case 2**: There are two errors in the first head which are a deletion in the i_1-th bit and a sticky insertion in the i_2-th bit but c_{i_1} and c_{i_2} are in the same run in $\boldsymbol{c}$. Then, in the second head, the bit c_{i_1+t} is deleted and the bit c_{i_2+t} is repeated.

- If c_{i_1+t} and c_{i_2+t} are also in the same run in $\boldsymbol{c}$, then $\boldsymbol{c}_2 = \boldsymbol{c}$. Thus $\boldsymbol{c}_1 = \boldsymbol{c}_2 = \boldsymbol{c}$.
- If c_{i_1+t} and c_{i_2+t} are not in the same run in $\boldsymbol{c}$, then $\boldsymbol{c}_2 \neq \boldsymbol{c}$. Thus $\boldsymbol{c}_1 \neq \boldsymbol{c}_2$. Hence, there exists j_1 such that $\boldsymbol{c}_1[j_1] \neq \boldsymbol{c}_2[j_1]$ and $\boldsymbol{c}_1[j] = \boldsymbol{c}_2[j]$ for all $j < j_1$. We now consider the following two cases

to prove that $j_1 \leqslant \min\{i_1 + t + t_1, i_2 + t + t_1\} < \min\{i_1 + 2t, i_2 + 2t\}$.

- Case 1: The first error is the deletion, that is, $i_1 < i_2$. If $\boldsymbol{c}_1[1, i_1+t+t_1] = \boldsymbol{c}_2[1, i_1+t+t_1]$ then the substring $\boldsymbol{c}[i_1 + t, i_1 + t + t_1]$ is a run of length $t_1 + 1$ since c_{i_1+t} and c_{i_2+t} are not in the same run. It is a contradiction as the length of any run of $\boldsymbol{c}$ is at most t_1. Therefore, $\boldsymbol{c}_1[1, i_1 + t + t_1] \neq \boldsymbol{c}_2[1, i_1 + t + t_1]$, that is, $j_1 \leqslant i_1 + t + t_1$.
- Case 2: The first error is the sticky insertion, that is, $i_1 > i_2$. Similarly as in the previous case, we can also show that $j_1 \leqslant i_2 + t + t_1$.

Let us consider the output of the third head, we see that $\boldsymbol{c}_3[1, j_1] = \boldsymbol{c}[1, j_1] = \boldsymbol{c}_1[1, j_1]$ since in the third head there is no error in first $\min\{i_1 + 2t, i_2 + 2t\} > j_1$ positions. Hence $\boldsymbol{c}_1[j_1] = \boldsymbol{c}_3[j_1] \neq \boldsymbol{c}_2[j_1]$.

Therefore, if $\boldsymbol{c} = \boldsymbol{c}_1$ then one of the above two conditions holds.

Now we assume that $\boldsymbol{c} \neq \boldsymbol{c}_1$. Then there are two errors in the first head which are a deletion in the i_1-th bit and a sticky insertion in the i_2-th bit but c_{i_1} and c_{i_2} are not in the same run in $\boldsymbol{c}$. Let j_1 be the leftmost index that $\boldsymbol{c}$ and $\boldsymbol{c}_1$ differ. Using the same argument as in the previous case, we obtain that j_1 exists and $j_1 \leqslant \min\{i_1+t_1, i_2+t_1\} < \min\{i_1+t, i_2+t\}$. Thus, $\boldsymbol{c}_1[j_1] \neq \boldsymbol{c}_2[j_1] = \boldsymbol{c}_3[j_1]$. Therefore, if $\boldsymbol{c} \neq \boldsymbol{c}_1$ then none of the claim's conditions holds, which verifies the correctness of the claim. ∎

If one of the two conditions in Claim 29 holds, then $\boldsymbol{c} = \boldsymbol{c}_1$, and so we can recover the stored codeword by just taking the first vector $\boldsymbol{c}_1$. Otherwise, there are one deletion and one sticky insertion in the first head. However, we do not know which error occurs first.

Step 2: When reaching this step we know that $\boldsymbol{c}_1 \neq \boldsymbol{c}$ and as a result of Claim 29 $\boldsymbol{c}_1 \neq \boldsymbol{c}_2$ and c_{i_1} and c_{i_2} are not in the same run of $\boldsymbol{c}$. Let j_1 be the leftmost index that $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ differ and let $\boldsymbol{c}'_1$ be the vector obtained by inserting $\boldsymbol{c}_2[j_1]$ into the j_1-th location of $\boldsymbol{c}_1$. We next present a condition which determines whether the first error occurred is a deletion or a sticky insertion.

Claim 30: The first position error is determined to be a deletion or a sticky insertion according to the following two decision rules:

1) *If* $\boldsymbol{c}'_1[1, j_1 + 2t_1 - 2] = \boldsymbol{c}_2[1, j_1 + 2t_1 - 2]$ *then the first error in the first head is a deletion.*
2) *If* $\boldsymbol{c}'_1[1, j_1 + 2t_1 - 2] \neq \boldsymbol{c}_2[1, j_1 + 2t_1 - 2]$ *then let* $\boldsymbol{c}''_1$ *be the vector obtained by deleting the* j_2*-th bit in* $\boldsymbol{c}'_1$ *in which* j_2 *is the leftmost index that* $\boldsymbol{c}'_1$ *and* $\boldsymbol{c}_2$ *differ.*
   - *If* $\boldsymbol{c}''_1[1, j_1 + 2t_1 - 2] = \boldsymbol{c}_2[1, j_1 + 2t_1 - 2]$ *then the original vector* $\boldsymbol{c} = \boldsymbol{c}''_1$.
   - *If* $\boldsymbol{c}''_1[1, j_1 + 2t_1 - 2] \neq \boldsymbol{c}_2[1, j_1 + 2t_1 - 2]$ *then the first error in the first head is a sticky insertion.*

Proof: We first describe the possible outcomes in case the first position error is a deletion or a sticky insertion. Based on these observations, we will then verify the correctness of the conditions in the claim.

1) **Case 1**: The first error is a deletion, that is, $i_1 < i_2$, so the outputs from the first two heads are:

$$\boldsymbol{c}_1 = (c_1, \ldots, c_{i_1-1}, c_{i_1+1}, \ldots, c_{i_2}, c_{i_2}, c_{i_2+1} \ldots, c_n),$$
$$\boldsymbol{c}_2 = (c_1, \ldots, c_{i_1-1}, c_{i_1}, \ldots, c_{i_1+t-1}, c_{i_1+t+1}, \ldots, c_{i_2+t}, c_{i_2+t}, c_{i_2+t} + 1, \ldots, c_n).$$

Then $\boldsymbol{c}'_1$ is the vector obtained after correcting the first deletion in the first head, that is,

$$\boldsymbol{c}'_1 = (c_1, \ldots, c_{i_1-1}, c_{i_1}, c_{i_1+1}, \ldots, c_{i_2}, c_{i_2}, c_{i_2+1} \ldots, c_n).$$

Note that since j_1 is the left most index that $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ differ, we have that

$$c_{i_1} = c_{i_1+1} = \cdots = c_{j_1-1} = c_{j_1} \tag{6}$$

and so inserting the bit $\boldsymbol{c}_2[j_1]$, which is the bit c_{j_1}, in the j_1-th position of $\boldsymbol{c}_1$ is equivalent to inserting the bit c_{i_1} in the i_1-th position of $\boldsymbol{c}_1$. According to (6), the subvector $\boldsymbol{c}[i_1, j_1]$ is a run and thus $j_1 < i_1 + t_1$. Therefore, we have that $j_1 + 2t_1 - 2 < i_1 + 3t_1 - 2 = i_1 + t$. Thus, $\boldsymbol{c}_2[1, j_1 + 2t_1 - 2] = \boldsymbol{c}[1, j_1 + 2t_1 - 2]$ since the first error happens in the second head at position $i_1 + t > j_1 + 2t_1 - 2$.

- If $\boldsymbol{c}'_1[1, j_1+2t_1-2] = \boldsymbol{c}_2[1, j_1+2t_1-2] = \boldsymbol{c}[1, j_1+2t_1-2]$ then we know that in the first $j_1 + 2t_1 - 2$ bits in the first head, i.e. in $\boldsymbol{c}_1$, there is only a single deletion which is corrected in $\boldsymbol{c}'_1$. In this case, the second error is at least $2t_1 - 2$ positions apart from the first error. That is, $i_2 - i_1 > 2t_1 - 2$.
- If $\boldsymbol{c}'_1[1, j_1+2t_1-2] \neq \boldsymbol{c}_2[1, j_1+2t_1-2] = \boldsymbol{c}[1, j_1+2t_1-2]$ then we know that the position of the second error in the first head, which is a sticky insertion, is within the first $j_1 + 2t_1 - 2$ bits, i.e., $i_2 \leqslant j_1 + 2t_1 - 2$. Therefore, $\boldsymbol{c}''_1$ is the vector obtained by correcting both of the errors. In this case

$$\boldsymbol{c}''_1[1, j_1 + 2t_1 - 2] = \boldsymbol{c}_2[1, j_1 + 2t_1 - 2] \tag{7}$$

and furthermore, $\boldsymbol{c}''_1$ provides us with the original codeword $\boldsymbol{c}$.

2) **Case 2:** The first error is a sticky insertion, that is, $i_2 < i_1$. Then, the outputs from the first two heads are:

$$\boldsymbol{c}_1 = (c_1, \ldots, c_{i_2}, c_{i_2}, \ldots, c_{i_1-1}, c_{i_1+1}, \ldots, c_n),$$
$$\boldsymbol{c}_2 = (c_1, \ldots, c_{i_2}, c_{i_2+1}, \ldots, c_{i_2+t}, c_{i_2+t}, \ldots, c_{i_1-1+t}, c_{i_1+1+t}, \ldots, c_n).$$

Here we have that $(c_{i_2}, c_{i_2+1}, \ldots, c_{j_1-1})$ is a run and therefore $j_1 \leqslant i_2 + t_1 < i_2 + t$, so there are no errors in the first j_1 bits of $\boldsymbol{c}_2$. Hence, $\boldsymbol{c}'_1[1, j_1 + 1] = (c_1, \ldots, c_{i_2}, c_{i_2}, \ldots, c_{j_1-2}, c_{j_1}, c_{j_1-1})$. Note that $j_1 \leqslant i_1$ since c_{i_1} and c_{i_2} are not in the same run in $\boldsymbol{c}$. We consider two more cases here.

- **Case 2.1:** Assume $\boldsymbol{c}[j_1 - 1, i_1 + 1] = (c_{j_1-1}, \ldots, c_{i_1+1})$ is a subvector with period 2. Then, $i_1 \leqslant j_1 + t_1 - 3$ since $\boldsymbol{c} \in \mathbb{C}(n, \leqslant 2, t_1)$. Furthermore, since $j_1 \leqslant i_2 + t_1$ we conclude that

$$i_1+t_1+1 \leqslant j_1+t_1-3+t_1+1 \leqslant i_2+t_1+2t_1-2 = i_2+t,$$

and therefore $\boldsymbol{c}_2[1, i_1+t_1+1] = \boldsymbol{c}[1, i_1+t_1+1]$. In this case, we claim that the following two subvectors,

$$\begin{aligned}\boldsymbol{c}'_1[1, i_1+t_1+1] = (&c_1, \ldots, c_{j_1-1}, c_{j_1}, c_{j_1-1},\\ &c_{j_1}, \ldots, c_{i_1-1}, c_{i_1+1}, \ldots, c_{i_1+t_1}),\\ \boldsymbol{c}_2[1, i_1+t_1+1] = (&c_1, \ldots, c_{j_1-1}, c_{j_1}, c_{j_1+1},\\ &c_{j_1+2}, \ldots, c_{i_1+1}, c_{i_1+2}, \ldots,\\ &c_{i_1+t_1+1})\end{aligned}$$

are not the same. Assume the contrary, then the subvector $\boldsymbol{c}[i_1+1, i_1+t_1+1]$ is a run of length t_1+1, which is a contradiction. Furthermore, since $i_1 + t_1 + 1 \leqslant j_1 + 2t_1 - 2$, we conclude that

$$\boldsymbol{c}'_1[1, j_1+2t_1-2] \neq \boldsymbol{c}_2[1, j_1+2t_1-2]. \quad (8)$$

Hence, if j_2 is the leftmost index that $\boldsymbol{c}'_1$ and $\boldsymbol{c}_2$ differ then $j_2 \leqslant i_1 + t_1 + 1$. Let us consider the vector $\boldsymbol{c}''_1$ obtained by deleting the j_2-th bit in $\boldsymbol{c}'_1$. We have $\boldsymbol{c}''_1[1, j_2-1] = \boldsymbol{c}'_1[1, j_2-1] = \boldsymbol{c}_2[1, j_2-1] = \boldsymbol{c}[1, j_2-1]$ since j_2 is the leftmost index that $\boldsymbol{c}'_1[1, i_1+t_1+1]$ and $\boldsymbol{c}_2[1, i_1+t_1+1]$ differ. Since we assumed here that $(c_{j_1-1}, \ldots, c_{i_1+1})$ is a subvector with period 2, we get that $\boldsymbol{c}'_1[1, i_1+1] = \boldsymbol{c}_2[1, i_1+1]$, and thus we deduce that $j_2 > i_1 + 1$, which provides that $\boldsymbol{c}''_1[j_2, n] = \boldsymbol{c}[j_2, n]$. Together we conclude that $\boldsymbol{c}''_1 = \boldsymbol{c}$ and in particular

$$\boldsymbol{c}''_1[1, j_1+2t_1-2] = \boldsymbol{c}_2[1, j_1+2t_1-2], \quad (9)$$

since $j_1 + 2t_1 - 2 \leqslant i_2 + t$. In this case, although our guess is wrong we can still recover the original vector.

- **Case 2.2:** Assume $\boldsymbol{c}[j_1-1, i_1+1] = (c_{j_1-1}, \ldots, c_{i_1+1})$ is not a subvector with period 2. First we deduce from this condition that $\boldsymbol{c}'_1[j_1+1, i_1+1] \neq \boldsymbol{c}_2[j_1+1, i_1+1]$. Furthermore, in this case we get that $\boldsymbol{c}'_1[j_1+1, j_1+t_1-1] \neq \boldsymbol{c}_2[j_1+1, j_1+t_1-1]$ since otherwise $\boldsymbol{c}[j_1-1, j_1+t_1-1]$ is a subvector with period 2 of length t_1+1, which is a contradiction. Thus,

$$\boldsymbol{c}'_1[1, j_1+2t_1-2] \neq \boldsymbol{c}_2[1, j_1+2t_1-2] \quad (10)$$

since $j_1+t_1-1 \leqslant j_1+2t_1-2$. So, if j_2 is the leftmost index that $\boldsymbol{c}'_1$ and $\boldsymbol{c}_2$ differ then $j_2 \leqslant \min\{j_1+t_1-1, i_1+1\}$.
Now, we consider the following two more cases:

  - **Case 2.2.1:** $\boldsymbol{c}[j_2-1, i_1] = (c_{j_2-1}, \ldots, c_{i_1})$ is a run of length $i_1 - j_2 + 2$. Then $i_1 \leqslant j_2 + t_1 - 2$. Since $j_2 \leqslant j_1 + t_1 - 1$ and $j_1 \leqslant i_2 + t_1$ we get that

$$\begin{aligned}i_1 &\leqslant j_2 + t_1 - 2 \leqslant j_1 + t_1 - 1 + t_1 - 2\\ &\leqslant i_2 + t_1 + 2t_1 - 3 < i_2 + t.\end{aligned}$$

    In this case, the following two subvectors

$$\begin{aligned}\boldsymbol{c}''_1[1, i_1] = (&c_1, \ldots, c_{j_1-1}, c_{j_1}, c_{j_1-1}, \ldots, c_{j_2-3},\\ &c_{j_2-1}, \ldots, c_{i_1-1}),\\ \boldsymbol{c}_2[1, i_1] = (&c_1, \ldots, c_{j_1-1}, c_{j_1}, c_{j_1+1}, \ldots, c_{j_2-1},\\ &c_{j_2}, \ldots, c_{i_1})\end{aligned}$$

    are the same. That is, $\boldsymbol{c}''_1[1, i_1] = \boldsymbol{c}[1, i_1]$ since in the second head, there is no error in first $i_1 < i_2 + t$ bits. Moreover, $\boldsymbol{c}''_1[i_1+1, n] = \boldsymbol{c}[i_1+1, n]$ since in the first head, there is no error in last $n - i_1 - 1$ bits. Therefore, $\boldsymbol{c}''_1 = \boldsymbol{c}$ and $\boldsymbol{c}''_1[1, j_1 + 2t_1 - 2] = \boldsymbol{c}_2[1, j_1 + 2t_1 - 2]$. We recovered the original vector already.
  - **Case 2.2.2:** $\boldsymbol{c}[j_2-1, i_1] = (c_{j_2-1}, \ldots, c_{i_1})$ is not a run. In this case since $j_2 + t_1 - 1 \leqslant j_1 + 2t_1 - 2 \leqslant i_2 + t$, we first have that $\boldsymbol{c}_2[1, j_2+t_1-1] = \boldsymbol{c}[1, j_2+t_1-1]$, that is, there are no errors in the first $j_2 + t_1 - 1$ bits of $\boldsymbol{c}_2$. Furthermore, $\boldsymbol{c}''_1[j_2-1, j_2+t_1-1] \neq \boldsymbol{c}_2[j_2-1, j_2+t_1-1] = \boldsymbol{c}[j_2-1, j_2+t_1-1]$ since otherwise there is a run of length t_1+1 in $\boldsymbol{c}$. Together we conclude that

$$\boldsymbol{c}''_1[1, j_1+2t_1-2] \neq \boldsymbol{c}_2[1, j_1+2t_1-2]. \quad (11)$$

    It is a contradiction since in first $i_2 + t$ bits, there is no error in the second head and there are at most 2 errors in the first head. Therefore, we know that our guess is wrong and the first error in the first head is actually the sticky insertion.

Now we are ready to prove the correctness of the two decision rules. According to (8) and (10) we see that if a sticky insertion occurs then $\boldsymbol{c}'_1[1, j_1+2t_1-2] \neq \boldsymbol{c}_2[1, j_1+2t_1-2]$. Therefore, in case there is equality between these two subvectors then the first position error is necessarily a deletion. As for the second decision rule, we first assume that $\boldsymbol{c}'_1[1, j_1+2t_1-2] \neq \boldsymbol{c}_2[1, j_1+2t_1-2]$. For the deletion case, according to (7), we see that if $\boldsymbol{c}''_1[1, j_1+2t_1-2] = \boldsymbol{c}_2[1, j_1+2t_1-2]$ then we successfully correct the original codeword $\boldsymbol{c}$ by the vector $\boldsymbol{c}''$. Similarly for the sticky insertion case, according to (9), we see that if $\boldsymbol{c}''_1[1, j_1+2t_1-2] = \boldsymbol{c}_2[1, j_1+2t_1-2]$, again we successfully correct the original codeword $\boldsymbol{c}$ to be $\boldsymbol{c}''$. Lastly, we assume that $\boldsymbol{c}'_1[1, j_1+2t_1-2] = \boldsymbol{c}_2[1, j_1+2t_1-2]$. Here, we showed in (7) that if a deletion occured and $\boldsymbol{c}'_1[1, j_1 + 2t_1 - 2] \neq \boldsymbol{c}_2[1, j_1 + 2t_1 - 2]$ then necessarily $\boldsymbol{c}''_1[1, j_1 + 2t_1 - 2] = \boldsymbol{c}_2[1, j_1 + 2t_1 - 2]$, which verifies this condition. ∎

After step 2, we are either able to recover the original vector or are able to determine whether a deletion or a sticky insertion occurs first.

Step 3: Now, we are ready to recover the original vector, in case it was not done in Step 2, where we know whether the first error is a deletion or a sticky insertion. We first consider the deletion case. According to Claim 30, in this case $\boldsymbol{c}'_1[1, j_1+2t_1-2] = \boldsymbol{c}_2[1, j_1+2t_1-2]$. Thus, we know that the second error is at least $2t_1 - 2$ positions apart from the first error, and this holds also for the second head. In this case we apply the same procedure for the second and third heads

to obtain the vector $\boldsymbol{c}'_2$ after correcting the first deletion in $\boldsymbol{c}_2$. Lastly, all we need is to correct a single sticky insertion in $\boldsymbol{c}'_1$ and $\boldsymbol{c}'_2$, which can be done by using Theorem 23.

The case of sticky insertion is proved in a similar way. First we remove the j_1-th bit from $\boldsymbol{c}_1$ to obtain the vector $\boldsymbol{c}'_1$. We apply the same rule on the second and third heads to obtain the vector $\boldsymbol{c}'_2$ that corrects the first sticky insertion in $\boldsymbol{c}_2$, and we complete the decoding task by correcting a single deletion in $\boldsymbol{c}'_1$ and $\boldsymbol{c}'_2$, which can be done by using Theorem 2. ■

We observe that the proof of Theorem 28 also provides an efficient decoding algorithm to recover the stored codeword in $\mathbb{C}_3(n, \leqslant 2, t_1)$ using three heads. The following example demonstrates this decoding procedure.

Example 8: Let $n = 20, t_1 = 3, t = 7$ and

$$\boldsymbol{c} = (1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0)$$

is a stored codeword in $\mathbb{C}_3(n, \leqslant 2, t_1)$. Assume that the outputs from the three heads are:

$$\textbf{Head 1}: \boldsymbol{c}_1 = \boldsymbol{c}(\gamma_2, \delta_5) = (1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0)$$

$$\textbf{Head 2}: \boldsymbol{c}_2 = \boldsymbol{c}(\gamma_9, \delta_{12}) = (1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0)$$

$$\textbf{Head 3}: \boldsymbol{c}_3 = \boldsymbol{c}(\gamma_{16}, \delta_{19}) = (1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0)$$

In Step 1, we see that the condition in Claim 30 does not hold since $\boldsymbol{c}_1[3] \neq \boldsymbol{c}_2[3] = \boldsymbol{c}_3[3]$, and therefore we conclude that $\boldsymbol{c} \neq \boldsymbol{c}_1$. In Step 2, we determine whether the first error is a deletion or a sticky insertion. It is easy to see that $j_1 = 3$ and thus we construct the vector

$$\boldsymbol{c}'_1 = (1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0).$$

Since $j_1 + 2t_1 - 2 = 7$ and $\boldsymbol{c}'_1[1, 7] \neq \boldsymbol{c}_2[1, 7]$, we find the leftmost index that $\boldsymbol{c}'_1$ and $\boldsymbol{c}_2$ differ, which is $j_2 = 4$. Hence, we construct the vector

$$\boldsymbol{c}''_1 = (1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0).$$

Since $\boldsymbol{c}''_1[1, 7] \neq \boldsymbol{c}_2[1, 7]$, we deduce that the first position error in the first head is a sticky insertion. In Step 3, we use the first two vectors $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ to correct the first sticky insertion in the first head to obtain the vector $\boldsymbol{c}(\delta_5) = (1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0)$. Then, we use the last two vectors $\boldsymbol{c}_2$ and $\boldsymbol{c}_3$ to correct the first sticky insertion in the second head to obtain the vector $\boldsymbol{c}(\delta_{12}) = (1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0)$. Lastly, we correct the last deletion to recover the stored codeword $\boldsymbol{c} = (1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0)$. □

The cardinality of the code $\mathbb{C}_3(n, \leqslant 2, t_1)$ is provided in Lemma 15. Hence, similarly to Corollary 16, we conclude with the following corollary.

Corollary 31: There exists a three-head two-position-error-correcting code with approximately $\log(e)/4 \approx 0.36$ *bits of redundancy when the distance between adjacent heads is at least* $t = 3(\lceil \log(n) \rceil + 2) - 2$.

Lastly, we present another construction with a decoding algorithm in case the distance between consecutive position errors is large enough. This will be proved in the next theorem.

Theorem 32: The code $\mathbb{C}_3(n, \leqslant 2, t_1)$ *is a two-head d-position-error-correcting codes for all d if the distance between any two consecutive heads is at least* $t = 2t_1 - 1$ *and the distance between any two consecutive position errors is at least* $2t$.

Proof: Let $\boldsymbol{c} = (c_1, \ldots, c_n) \in \mathbb{C}_3(n, \leqslant 2, t_1)$ be a stored codeword. Assume that the index set $\mathcal{I} = \{i_1, i_2, \ldots, i_r\}$ in which $i_1 < i_2 < \cdots < i_r$ is the set of r locations where the position errors occur in the first head. We also assume here that for all $\ell \in [r-1]$, $i_{\ell+1} - i_\ell \geqslant 2t$. The set of locations where the position errors occur in the second head is given by $\mathcal{I} + t = \{i_1 + t, i_2 + t, \ldots, i_r + t\}$. As before, $\boldsymbol{c}_1, \boldsymbol{c}_2$ is the read vector by the first, second head respectively.

If the first error is a deletion then the first $i_1 + t_1 - 1$ bits read by the two heads are

$$\boldsymbol{c}_1[1, i_1 + t_1 - 1] = (c_1, \ldots, c_{i_1-1}, c_{i_1+1}, c_{i_1+2} \ldots, c_{i_1+t_1}),$$
$$\boldsymbol{c}_2[1, i_1 + t_1 - 1] = (c_1, \ldots, c_{i_1-1}, c_{i_1}, c_{i_1+1}, \ldots, c_{i_1+t_1-1}).$$

We claim that $\boldsymbol{c}_1[1, i_1 + t_1 - 1] \neq \boldsymbol{c}_2[1, i_1 + t_1 - 1]$. Otherwise, we get that $(c_{i_1}, \ldots, c_{i_1+t_1})$ is a run of length $t_1 + 1$, which results with a contradiction since $\boldsymbol{c} \in \mathbb{C}_3(n, \leqslant 2, t_1)$.

In a similar way it is possible to show that if the first error is a sticky insertion then again $\boldsymbol{c}_1[1, i_1 + t_1 - 1] \neq \boldsymbol{c}_2[1, i_1 + t_1 - 1]$. Let j_1 be the leftmost index that the vectors $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ differ, so $j_1 \leqslant i_1 + t_1 - 1$.

Since we do not know whether the first position error is a deletion or a sticky insertion, we guess that it is a deletion and correct this error by inserting the j_1-th bit of the second vector $\boldsymbol{c}_2$ into the j_1-th position of the first vector $\boldsymbol{c}_1$, and we denote this vector by $\boldsymbol{c}'_1$.

If the first position error is indeed a deletion, that is, we guess correctly, then the first $j_1 + t_1 - 1$ bits in the two vectors $\boldsymbol{c}'_1$ and $\boldsymbol{c}_2$ are the same, i.e., $\boldsymbol{c}'_1[1, j_1 + t_1 - 1] = \boldsymbol{c}_2[1, j_1 + t_1 - 1]$. However, if the first error is a sticky insertion, that is, we did not guess correctly, then the first $j_1 + t - 1$ bits in the two vectors $\boldsymbol{c}'_1$ and $\boldsymbol{c}_2$ become:

$$\boldsymbol{c}'_1[1, j_1 + t_1 - 1] = (c_1, \ldots, c_{i_1}, c_{i_1}, \ldots, c_{j_1-2}, c_{j_1}, c_{j_1-1}, c_{j_1} \ldots, c_{j_1+t_1-3})$$
$$\boldsymbol{c}_2[1, j_1 + t_1 - 1] = (c_1, \ldots, c_{i_1}, c_{i_1+1}, \ldots, c_{j_1-1}, c_{j_1}, c_{j_1+1}, c_{j_1+2} \ldots, c_{j_1+t_1-1}).$$

Note that there are no new position errors within the first $j_1 + t_1 - 1$ bits since $j_1 + t_1 - 1 \leqslant i_1 + t - 1$. Next we also get that $\boldsymbol{c}'_1[j_1 + 1, j_1 + t_1 - 1] \neq \boldsymbol{c}_2[j_1 + 1, j_1 + t_1 - 1]$, since otherwise $(c_{j_1-1}, c_{j_1} \ldots, c_{j_1+t_1-1})$ is a vector with period 2 of length $t_1 + 1$. Therefore, $\boldsymbol{c}'_1[1, j_1 + t_1 - 1] \neq \boldsymbol{c}_2[1, j_1 + t_1 - 1]$ and we know that our guess is wrong and the position error is a sticky insertion. To correct this sticky insertion, we only need to remove the j_1-th bit in the first vector $\boldsymbol{c}_1$.

To conclude, we can always determine whether the first position error is a deletion or a sticky insertion and then correct it as described above. Note that this is possible to do since the first error in the second head and the second error in the first head are both far apart. We then repeat this procedure to correct the rest of the position errors. ■

VIII. Codes Correcting Combination of Substitution Errors and Position Errors

So far in this paper we only discussed position errors of insertions and deletions. In this section, we consider also the case in which there are also substitution errors and construct a code correcting substitution and position errors.

For a length-n word $\boldsymbol{u} = (u_1, \ldots, u_n) \in \mathbb{F}_2^n$ and $i \in [n]$, let $\overline{u}_i = 1 - u_i$ be the inverse of bit u_i and $\overleftarrow{\boldsymbol{u}} = (u_n, \ldots, u_1)$ be the reverse of the vector $\boldsymbol{u}$. Let us denote by $\boldsymbol{u}(\epsilon_i)$ the vector obtained from a substitution error of the i-th bit u_i, that is, $\boldsymbol{u}(\epsilon_i) = (u_1, \ldots, \overline{u}_i, \ldots, u_n)$. In case there are a substitution error of the i_1-th bit and a deletion (or sticky insertion) error of the i_2-th bit, we obtain the vector $\boldsymbol{u}(\epsilon_{i_1}, \delta_{i_2})$ (or $\boldsymbol{u}(\epsilon_{i_1}, \gamma_{i_2})$). That is, if $i_1 < i_2$ then

$$\boldsymbol{u}(\epsilon_{i_1}, \delta_{i_2}) = (u_1, \ldots, \overline{u}_{i_1}, \ldots, u_{i_2-1}, u_{i_2+1}, \ldots, u_n).$$

We note that in our model of two heads at distance t, if an error occurs at the i-th position in the first head then a corresponding error also occurs at the $(i+t)$-th position in the second head. That is, if $\boldsymbol{u}$ is the original codeword and the output from the first head is $\boldsymbol{u}(\epsilon_{i_1}, \delta_{i_2})$, then the output from the second head is $\boldsymbol{u}(\epsilon_{i_1+t}, \delta_{i_2+t})$.

Recall that our first main result in this paper shows that the code $\mathbb{C}_1(n, 1, t_1)$, which could be encoded with only a single bit of redundancy, can correct a single deletion using two heads of distance at least t_1. In this section, we show that this code is also capable of correct a combination of a single substitution error and a single position error using two heads at distance at least $t = 3t_1 + 1$. Note that while a substitution error does not change the length of the codeword, a deletion (or a sticky insertion) decreases (or increases) the length of the codeword by one. Thus, if there are a single substitution error and a single position error, we can determine whether that position error is a deletion or a sticky insertion by the length of the obtained vector. This result is proved in the next theorem.

Theorem 33: The code $\mathbb{C}_1(n, 1, t_1)$ can correct a single substitution error and a single position error using two heads at distance $t = 3t_1 + 1$.

Proof: We prove the statement in the theorem only for the case where the position error is a deletion error since the proof for a sticky insertion repeats the same ideas.

Let $\boldsymbol{c} = (c_1, \ldots, c_n) \in \mathbb{C}_1(n, 1, t_1)$ be the stored codeword and let $\boldsymbol{c}_1, \boldsymbol{c}_2 \in \{0, 1\}^{n-1}$ be the received words from the first, second head, respectively. In case there is a substitution error of the i_1-th bit and a deletion error of the i_2-th bit, the output of the first head is $\boldsymbol{c}_1 = \boldsymbol{c}(\epsilon_{i_1}, \delta_{i_2})$ and the output of the second head is $\boldsymbol{c}_2 = \boldsymbol{c}(\epsilon_{i_1+t}, \delta_{i_2+t})$, where $t = 3t_1 + 1$ is the distance between the two heads. To prove the theorem, we show how to recover the stored codeword $\boldsymbol{c}$ from the two vectors $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$. Note that if we know whether $i_1 < i_2$ or $i_1 > i_2$ then we can recover the stored codeword, as will be shown in Claim 34. However, in our model, we do not have this information. Hence, Claim 35 shows how to determine whether $i_1 < i_2$ or $i_1 > i_2$ and thus it is possible to recover the stored codeword, which will complete the proof.

Algorithm 1 $\texttt{decode}(\boldsymbol{c}_1, \boldsymbol{c}_2)$

Input: $\boldsymbol{c}_1, \boldsymbol{c}_2 \in \mathbb{F}_2^{n-1}$.
Output: $\boldsymbol{c} \in \mathbb{F}_2^n$ or 0.

1. $j_1 \leftarrow$ leftmost index that $\boldsymbol{c}_1 \neq \boldsymbol{c}_2$;
2. $\boldsymbol{c}'_1 \leftarrow \boldsymbol{c}_1(\epsilon_{j_1})$;
3. $\boldsymbol{c}'_2 \leftarrow \boldsymbol{c}_2(\epsilon_{j_1+t})$;
4. $j_2 \leftarrow$ leftmost index that $\boldsymbol{c}'_1 \neq \boldsymbol{c}'_2$;
5. $\boldsymbol{c}''_1 \leftarrow \boldsymbol{c}'_2[1, j_2] \circ \boldsymbol{c}'_1[j_2, n-1]$;
6. $j_3 \leftarrow$ rightmost index that $\boldsymbol{c}'_1 \neq \boldsymbol{c}'_2$;
7. $\boldsymbol{c}''_2 \leftarrow \boldsymbol{c}'_2[1, j_3] \circ \boldsymbol{c}'_1[j_3, n-1]$;

if $\boldsymbol{c}''_1 = \boldsymbol{c}''_2$ **then**
  return $\boldsymbol{c} = \boldsymbol{c}''_1$;
else
  return 0;
end if

Claim 34: If $i_1 < i_2$ then Algorithm 1 successfully recovers the stored codeword $\boldsymbol{c}$, that is, $\texttt{decode}(\boldsymbol{c}_1, \boldsymbol{c}_2) = \boldsymbol{c}$.

Proof: It $i_1 < i_2$ then the two received vectors are:

$$\begin{aligned}
\textbf{Head 1}: \boldsymbol{c}_1 &= \boldsymbol{c}(\epsilon_{i_1}, \delta_{i_2}) \\
&= (c_1, \ldots, \overline{c}_{i_1}, \ldots, c_{i_2-1}, c_{i_2+1}, \ldots, c_n), \\
\textbf{Head 2}: \boldsymbol{c}_2 &= \boldsymbol{c}(\epsilon_{i_1+t}, \delta_{i_2+t}) \\
&= (c_1, \ldots, \overline{c}_{i_1+t}, \ldots, c_{i_2+t-1}, c_{i_2+t+1}, \ldots, c_n).
\end{aligned}$$

Since $\boldsymbol{c}_1[1, i_1 - 1] = \boldsymbol{c}_2[1, i_1 - 1]$ and $\overline{c}_{i_1} \neq c_{i_1}$, the leftmost index such that $\boldsymbol{c}_1 \neq \boldsymbol{c}_2$ is i_1, that is $j_1 = i_1$ in Step 1 in Algorithm 1. Since the first error is a substitution, we correct this substitution in both heads to obtain $\boldsymbol{c}'_1 = \boldsymbol{c}_1(\epsilon_{j_1}) = \boldsymbol{c}(\delta_{i_2})$ and $\boldsymbol{c}'_2 = \boldsymbol{c}_2(\epsilon_{j_1+t}) = \boldsymbol{c}(\delta_{i_2+t})$ in Steps 2 and 3. Next we follow the proof of Theorem 2 in Steps 4 and 5 in order to get $\boldsymbol{c} = \boldsymbol{c}''_1$. Similarly, we obtain $\boldsymbol{c}''_2 = \boldsymbol{c}$. Therefore, the output of Algorithm 1 is the stored codeword $\boldsymbol{c}$. ■

Claim 35: If the output of Algorithm 1 is a vector $\boldsymbol{c} \in \mathbb{F}_2^n$ then $i_1 < i_2$ and $\boldsymbol{c}$ is the stored codeword. Otherwise, that is, $\texttt{decode}(\boldsymbol{c}_1, \boldsymbol{c}_2) = 0$, then $i_1 > i_2$.

Proof: Recall that according to Claim 34 if $i_1 < i_2$ then the output of Algorithm 1 is $\texttt{decode}(\boldsymbol{c}_1, \boldsymbol{c}_2) = \boldsymbol{c}$. We now consider the case where $i_1 > i_2$ and show that in this case the output of Algorithm 1 is 0, that is $\texttt{decode}(\boldsymbol{c}_1, \boldsymbol{c}_2) = 0$, or $\boldsymbol{c}_1 \neq \boldsymbol{c}_2$. In this case, the two received vectors are:

$$\begin{aligned}
\textbf{Head 1}: \boldsymbol{c}_1 &= \boldsymbol{c}(\epsilon_{i_1}, \delta_{i_2}) \\
&= (c_1, \ldots, c_{i_2-1}, c_{i_2+1}, \ldots, \overline{c}_{i_1}, \ldots, c_n), \\
\textbf{Head 2}: \boldsymbol{c}_2 &= \boldsymbol{c}(\epsilon_{i_1+t}, \delta_{i_2+t}) \\
&= (c_1, \ldots, c_{i_2+t-1}, c_{i_2+t+1}, \ldots, \overline{c}_{i_1+t}, \ldots, c_n).
\end{aligned}$$

First observe that $\boldsymbol{c}_1[i_1 + t - 1] = c_{i_1+t} \neq \overline{c}_{i_1+t} = \boldsymbol{c}_2[i_1 + t - 1]$. Hence $\boldsymbol{c}_1 \neq \boldsymbol{c}_2$, that is, the index j_1 in Step 1 indeed exists. Moreover, $j_1 \geqslant i_2$ since $\boldsymbol{c}_1[1, i_2 - 1] = \boldsymbol{c}_2[1, i_2 - 1]$. However, j_1 may or may not be larger than $i_1 - 1$. We consider both of these cases here.

- **Case 1** ($j_1 < i_1 - 1$)**:** In this case, $j_1 \leqslant i_2 + t_1 - 1$. Otherwise, $(c_{i_2}, \ldots, c_{i_2+t_1})$ is a run of length t_1+1 which is a contradiction. In Steps 2 and 3, we changed the bit

$\boldsymbol{c}_1[j_1]$ in the vector $\boldsymbol{c}_1$ and the bit $\boldsymbol{c}_2[j_1+t]$ in the vector $\boldsymbol{c}_2$ to obtain the two vectors $\boldsymbol{c}'_1$ and $\boldsymbol{c}'_2$ as follows:

$$\begin{aligned}\boldsymbol{c}'_1 &= \boldsymbol{c}_1(\epsilon_{j_1})\\ &= (c_1,\ldots,c_{i_2-1},c_{i_2+1},\ldots,\overline{c}_{j_1+1},\ldots,\overline{c}_{i_1},\ldots,c_n),\\ \boldsymbol{c}'_2 &= \boldsymbol{c}_2(\epsilon_{j_1+t})\\ &= (c_1,\ldots,c_{i_2+t-1},c_{i_2+t+1},\ldots,\overline{c}_{j_1+t+1},\ldots,\\ &\quad \overline{c}_{i_1+t},\ldots,c_n).\end{aligned}$$

We again observe that $\boldsymbol{c}'_1 \neq \boldsymbol{c}'_2$ since $\boldsymbol{c}'_1[i_1+t-1] = c_{i_1+t} \neq \overline{c}_{i_1+t} = \boldsymbol{c}'_2[i_1+t-1]$ and thus the indices j_2 and j_3 in Steps 4 and 6 exist. We can also determine that $j_3 = i_1+t-1$ since $\boldsymbol{c}'_1[i_1+t,n-1] = \boldsymbol{c}'_2[i_1+t,n-1] = (c_{i_1+t+1},\ldots,c_n)$ and $\boldsymbol{c}'_1[i_1+t-1] \neq \boldsymbol{c}'_2[i_1+t-1]$. However, we do not know whether $j_2 \geqslant i_1$ or not. Hence, we consider these two cases here.

- **Case 1.1** ($j_2 < i_1$)**:** In this case, $j_2 \leqslant j_1+t_1$. Otherwise, $(c_{j_1+1},\ldots,c_{j_1+t_1})$ is a run of length t_1+1, which is a contradiction. Hence, $j_2 \leqslant j_1+t_1 \leqslant i_2+2t_1-1$. The concatenations in Steps 5 and 7 provide the following two vectors $\boldsymbol{c}''_1$ and $\boldsymbol{c}''_2$:

  $$\begin{aligned}\boldsymbol{c}''_1 &= (c_1,\ldots,c_{j_2},c_{j_2+1},\ldots,\overline{c}_{i_1},\ldots,c_n),\\ \boldsymbol{c}''_2 &= (c_1,\ldots,c_{i_2+t-1},c_{i_2+t+1},\ldots,\overline{c}_{j_1+t+1},\ldots,\\ &\quad \overline{c_{i_1+t}},c_{i_1+t},\ldots,c_n).\end{aligned}$$

  Now we prove that $\boldsymbol{c}''_1 \neq \boldsymbol{c}''_2$ by considering all three possibilities of i_1 as follows.

  - **Case 1.1.1** ($i_1 < i_2+t$)**:** Observe that $\boldsymbol{c}''_1[i_1] = \overline{c}_{i_1} \neq c_{i_1} = \boldsymbol{c}''_2[i_1]$. Hence, $\boldsymbol{c}''_1 \neq \boldsymbol{c}''_2$.
  - **Case 1.1.2** ($i_2+t \leqslant i_1 \leqslant j_1+t$)**:** Let us consider the two following subvectors:

    $$\begin{aligned}\boldsymbol{c}''_1[i_1,i_1+t-1] &= (\overline{c}_{i_1},\ldots,c_{i_1+t-1}),\\ \boldsymbol{c}''_2[i_1,i_1+t-1] &= (c_{i_1+1},\ldots,\overline{c}_{j_1+t+1},\ldots,\overline{c}_{i_1+t}).\end{aligned}$$

    If $\boldsymbol{c}''_1[i_1,i_1+t-1] = \boldsymbol{c}''_2[i_1,i_1+t-1]$ then $(c_{i_1+1},\ldots,c_{j_1+t})$, $(c_{j_1+t+1},\ldots,c_{i_1+t-1})$ is a run of length j_1+t-i_1, i_1-j_1-1, respectively. Hence, $j_1+t-i_1 \leqslant t_1$ and $i_1-j_1-1 \leqslant t_1$ and thus $t-1 \leqslant 2t_1$, which is a contradiction since $t = 3t_1 > 2t_1+1$. Therefore, $\boldsymbol{c}''_1 \neq \boldsymbol{c}''_2$.
  - **Case 1.1.3** ($i_1 \geqslant j_1+t+1$)**:** We observe that $\boldsymbol{c}''_1[i_1,i_1+t-1] \neq \boldsymbol{c}''_2[i_1,i_1+t-1]$, where

    $$\begin{aligned}\boldsymbol{c}''_1[i_1,i_1+t-1] &= (\overline{c}_{i_1},\ldots,c_{i_1+t-1},c_{i_1+t-1}),\\ \boldsymbol{c}''_2[i_1,i_1+t-1] &= (c_{i_1+1},\ldots,c_{i_1+t-1},\overline{c}_{i_1+t}).\end{aligned}$$

    Since the length-$(t-1)$ subvector $(c_{i_1+1},\ldots,c_{i_1+t-1})$ could not be a run, we get again that $\boldsymbol{c}''_1 \neq \boldsymbol{c}''_2$.

  Thus, the output of Algorithm 1 is zero in this case.
- **Case 1.2** ($j_2 \geqslant i_1$)**:** In this case, $i_1 \leqslant j_1+t_1+1 \leqslant i_2+2t_1 < i_2+t$. Otherwise, $(c_{j_1+1},\ldots,c_{j_1+t_1+1})$ is a run of length t_1+1, which is a contradiction. Furthermore, $j_2 < i_2+t$. Otherwise, $(c_{i_1},\ldots,c_{i_2+t})$ is a run of length i_2+t-i_1 since $\boldsymbol{c}'_1[i_1,i_2+t-1] = \boldsymbol{c}_2[i_1,i_2+t-1]$. It is a contradiction since $i_2+t-i_1 \geqslant (i_2+3t_1+1)-(i_2+2t_1) \geqslant t_1+1$. Therefore, $\boldsymbol{c}'_2[1,j_2] = \boldsymbol{c}[1,j_2] = (c_1,\ldots,c_{j_2})$. Moreover, $\boldsymbol{c}'_1[j_2,n-1] = \boldsymbol{c}[j_2+1,n] = (c_{j_2+1},\ldots,c_n)$ since $j_2 \geqslant i_1$. Concatenating as in Algorithm 1, we obtain the vector $\boldsymbol{c}''_1 = \boldsymbol{c} = (c_1,\ldots,c_{j_2},c_{j_2+1},\ldots,c_n)$. Therefore, if $\boldsymbol{c}''_2 = \boldsymbol{c}''_1$ then the output is the stored codeword $\boldsymbol{c}$ and if $\boldsymbol{c}''_2 \neq \boldsymbol{c}''_1$, then the output is zero.
- **Case 2** ($j_1 \geqslant i_1-1$)**:** In this case, $i_1 \leqslant i_2+t_1$. Otherwise, $(c_{i_2},\ldots,c_{i_2+t_1})$ is a run of length t_1+1 which is a contradiction. Furthermore, $j_1 \leqslant i_1+t_1$. Otherwise, $(c_{i_1},\ldots,c_{i_1+t_1})$ is a run of length t_1+1 which is again a contradiction. Hence, $j_1 \leqslant i_2+2t_1$ and thus $j_1+t_1 < i_2+t$. The two obtained vectors $\boldsymbol{c}'_1$ and $\boldsymbol{c}'_2$ are:

  $$\begin{aligned}\boldsymbol{c}'_1 &= \boldsymbol{c}_1(\epsilon_{j_1})\\ &= (c_1,\ldots,c_{i_2-1},c_{i_2+1},\ldots,\overline{c}_{i_1},\ldots,\overline{c}_{j_1+1},\ldots,c_n),\\ \boldsymbol{c}'_2 &= \boldsymbol{c}_2(\epsilon_{j_1+t})\\ &= (c_1,\ldots,c_{i_2+t-1},c_{i_2+t+1},\ldots,\overline{c}_{i_1+t},\ldots,\\ &\quad \overline{c}_{j_1+t+1},\ldots,c_n).\end{aligned}$$

  We observe that $\boldsymbol{c}'_1[j_1+1,i_2+t-1] \neq \boldsymbol{c}'_2[j_1+1,i_2+t-1]$. Otherwise, $(c_{j_1+1},\ldots,c_{i_2+t})$ is a run of length $i_2+t-j_1 > t_1$ which is a contradiction. Hence, the index j_2 in Step 4 exists and $j_2 \leqslant i_2+t-1$. Similarly to Case 1.2, $\boldsymbol{c}''_1 = \boldsymbol{c} = (c_1,\ldots,c_{j_2},c_{j_2+1},\ldots,c_n)$. Thus, if $\boldsymbol{c}''_2 = \boldsymbol{c}''_1$ then the output is the stored codeword $\boldsymbol{c}$ and if $\boldsymbol{c}''_2 \neq \boldsymbol{c}''_1$, then the output is zero.

After considering all possible outcomes, we conclude that if $\boldsymbol{c}''_1 = \boldsymbol{c}''_2$, that is, the algorithm's output is the vector $\boldsymbol{c}''_1$, then the output is the stored codeword and if $\boldsymbol{c}''_1 \neq \boldsymbol{c}''_2$, that is, the algorithm's output is 0 (zero), then $i_1 > i_2$. In this case, Claim 34 shows how to recover the stored codeword. ■

In conclusion, all we need to do in order to decode is apply Algorithm 1 on the vectors $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$. If the output is not 0 then the stored codeword output is $\texttt{decode}(\boldsymbol{c}_1,\boldsymbol{c}_2)$ and otherwise the correct output is $\overleftarrow{\boldsymbol{c}'}$, where $\boldsymbol{c}' = \texttt{decode}(\overleftarrow{\boldsymbol{c}_1},\overleftarrow{\boldsymbol{c}_2})$. Therefore, the code $\mathbb{C}_1(n,1,t_1)$ can correct a single substitution error and a single deletion error with two heads of distance at least $t = 3t_1+1$. ■

The following example demosntrates the decoding procedure of Algorithm 1.

Example 9: Let $n = 14$, $t = 6$ and

$$\boldsymbol{c} = (0,0,1,1,0,1,1,0,1,1,1,0,0,1)$$

be the stored codeword. Assume that in the first head, there are a substitution in the 3rd bit and a deletion of the 5th bit. Therefore, in the second head, there are a substitution in the 9th bit and a deletion of the 11th bit. The received word in the first, second head is:

$$\begin{aligned}\boldsymbol{c}_1 &= (0,0,0,1,1,1,0,1,1,1,0,0,1),\\ \boldsymbol{c}_2 &= (0,0,1,1,0,1,1,0,0,1,0,0,1),\end{aligned}$$

respectively. By comparing $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$, we see that the leftmost index that $\boldsymbol{c}_1 \neq \boldsymbol{c}_2$ is $j_1 = 3$. Hence, we obtain that

$$\begin{aligned}\boldsymbol{c}'_1 &= \boldsymbol{c}_1(\epsilon_3) = (0,0,1,1,1,1,0,1,1,1,0,0,1),\\ \boldsymbol{c}'_2 &= \boldsymbol{c}_2(\epsilon_9) = (0,0,1,1,0,1,1,0,1,1,0,0,1).\end{aligned}$$

By comparing $\boldsymbol{c}'_1$ and $\boldsymbol{c}'_2$, we see that $j_2 = 5$ is the leftmost index such that $\boldsymbol{c}'_1 \neq \boldsymbol{c}'_2$ and $j_3 = 8$ is the rightmost index that $\boldsymbol{c}'_1 \neq \boldsymbol{c}'_2$. Hence, we obtain

$$\boldsymbol{c}''_1 = \boldsymbol{c}'_2[1,5] \circ \boldsymbol{c}'_1[5,13] = (0,0,1,1,0,1,1,0,1,1,1,0,0,1),$$
$$\boldsymbol{c}''_2 = \boldsymbol{c}'_1[1,8] \circ \boldsymbol{c}'_2[8,13] = (0,0,1,1,0,1,1,0,1,1,1,0,0,1).$$

Lastly we get that $\boldsymbol{c}''_1 = \boldsymbol{c}''_2$, and thus we obtain $\boldsymbol{c} = \boldsymbol{c}''_2 = (0,0,1,1,0,1,1,0,1,1,1,0,0,1)$. □

IX. Connections to the Reconstruction Problem

Thus far in our work, we proposed a new model and coding solutions to combat position errors in racetrack memories. As mentioned in Section I, this model falls under the framework of the reconstruction problem which was first studied by Levenshtein [12], [13], and in particular for the cases of deletions and insertions. In Levenshtein's work [12], [13], the main problem was to study the minimum number of distinct erroneous versions which are needed in order to reconstruct the original vector, while proposing efficient reconstruction algorithms. While Levenshtein studied, for insertions and deletions, the case of uncoded vectors, recently Sala *et al.* [20] extended this study for the more general case where the transmitted vector belongs to some code with a prescribed minimum edit distance, and a similar study was carried for deletions in [7]. We also note that in the reconstruction problem, the assumption is that all received vectors are distinct. However, in our model of racetrack memories, we cannot have this assumption and hence we proposed a scheme using constrained codes and the positions of the heads in order to guarantee that all received vectors are distinct.

Let us consider for the rest of this section the case of a single deletion which was studied in Section III. Our result showed how to decode the stored codeword by using two heads, with at most a single bit of redundancy. On the other hand, if we could have the same assumption as in the reconstruction problem that the two outputs are different then it is possible to show that no coding is necessary in order to decode the deletion in each head. This can be accomplished since we have the knowledge that the deletion in the first head occurred before the deletion in the second head. However, this knowledge is not given in the reconstruction problem. In fact, in order to correct a single deletion in each output the result by Levenshtein claims that 3 different outputs are necessary and sufficient [12], [13]. Thus, we are interested in this section to study the reconstruction problem when there are only two different outputs. That is, each output has a single deletion while we don't know which output experienced the deletion first. Hence, it is necessary to use a coding solution in order to successfully decode the codeword. Note that a naive solution is to use a code that can correct a single deletion so its redundancy will be roughly $\log(n)$ bits, however this code can correct the deletion from any of the two outputs, and we are interested in better constructions that take advantage of the two different outputs. The next construction solves this problem, while using the shifted VT codes which were proposed recently in [23] when correcting bursts of deletions.

Construction 36: Given positive integers n, P, and nonnegative integers $0 \leqslant a < P, 0 \leqslant b < 2$, let the shifted Varshamov-Tenengolts (SVT) code be:

$$SVT_{a,b}(n,P) = \left\{ \boldsymbol{c} \in \mathbb{F}_2^n : \sum_{i=1}^{n} i \cdot c_i \equiv a (\mathrm{mod}\, P), \sum_{i=1}^{n} c_i \equiv b (\mathrm{mod} 2) \right\}.$$

Then,

$$\mathbb{C}(n,a,b,P) = \mathbb{C}_2(n,2,P) \cap SVT_{a,b}(n,P).$$

We state our contribution for the classical reconstruction problem in the following theorem and defer the proof to Appendix.

Theorem 37: Any codeword $\boldsymbol{c} \in \mathbb{C}(n,a,b,P)$ can be decoded from two different outputs that experienced each a single deletion.

The result in Theorem 37 holds also for the case of a single insertion in each output. The following corollary summarizes the redundancy result of this construction.

Corollary 38: There exists a code with at most $\log(\lceil \log n \rceil + 2) + 2$ *redundancy bits such that each codeword can be decoded from two different outputs that experienced each a single deletion (insertion).*

Proof: There exist $0 \leqslant a < P, 0 \leqslant b < 2$ such that the number of redundancy bits of the code $SVT_{a,b}(n,P)$ is at most $\log P + 1$ [23]. Let us choose $P = \lceil \log n \rceil + 2$, and according to Corollary 10, the code $\mathbb{C}_2(n,2,P)$ has at most a single bit of redundancy. Hence, according to the pigeonhole principle there exist $0 \leqslant a < P, 0 \leqslant b < 2$ such that the number of redundancy bits of the code $\mathbb{C}(n,a,b,P)$ is at most $\log P + 2$, when $P = \lceil \log n \rceil + 2$. ■

To conclude, we hope that this result will provide more knowledge on the reconstruction problem, and in particular on the trade-off between the number of redundancy bits and the number of required different outputs. It also shows a potential approach to construct codes to reconstruct the codeword with a fewer number of outputs compared to the worst-case.

X. Conclusion

In this work, we studied codes correcting position errors in racetrack memory. These errors were modeled as deletions and sticky insertions and the main goal of the paper is to use multiple read heads in order to correct the position errors. We first provided an explicit construction of two-head single-deletion-correcting code with approximately 0.36 redundancy bits. We then extended this construction for codes correcting burst of deletions and multiple deletions. Next, we investigated codes correcting multiple bursts of sticky insertions and any combination of two deletions and sticky insertions. Lastly, we extended our constructions for codes correcting a combination of substitutions and position errors and concluded with a construction in the classical reconstruction problem. While the constructions in the paper provide several solutions for codes correcting deletions and sticky insertions in racetrack memory, there are still several interesting problems which remain to be solved. These problems include in particular the construction of codes correcting multiple position errors which can be both

deletions and sticky insertions, and the study of other errors such as substitution errors.

APPENDIX
PROOF OF THEOREM 37

Let $\boldsymbol{c} = (c_1, \ldots, c_n)$ be a codeword in $\mathbb{C}(n, a, b, P)$ and $\boldsymbol{c}_1, \boldsymbol{c}_2$ be two received vectors. Let c_i, c_j be the deleted bit in $\boldsymbol{c}_1, \boldsymbol{c}_2$, respectively. That is, $\boldsymbol{c}_1 = (c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_n)$ and $\boldsymbol{c}_2 = (c_1, \ldots, c_{j-1}, c_{j+1}, \ldots, c_n)$. We now prove Theorem 37 by showing how to recover the original vector $\boldsymbol{c}$ explicitly in the following two steps:

- **Step 1:** Determine whether $i < j$ or $i > j$.
- **Step 2:** Correct a deletion to recover the original vector.

In the first step, the following claim provides a condition to determine whether $i < j$ or $i > j$.

Claim 39: Let j_1, j_2 be the leftmost, rightmost index such that $\boldsymbol{c}_1[j_1] \neq \boldsymbol{c}_2[j_1], \boldsymbol{c}_1[j_2] \neq \boldsymbol{c}_2[j_2]$, respectively. Let $\boldsymbol{c}_1^*$ be the vector obtained by inserting the bit $\boldsymbol{c}_2[j_1]$ into the j_1-th position in $\boldsymbol{c}_1$ and $\boldsymbol{c}_2^*$ be the vector obtained by inserting the bit $\boldsymbol{c}_1[j_2]$ into the j_2-th position in $\boldsymbol{c}_2$.

- *If $\boldsymbol{c}_1^* \neq \boldsymbol{c}_2^*$ then $i > j$.*
- *If $\boldsymbol{c}_1^* = \boldsymbol{c}_2^*$ and $\boldsymbol{c}_1^* \notin \mathbb{C}(n, a, b, P)$ then $i > j$.*
- *If $\boldsymbol{c}_1^* = \boldsymbol{c}_2^*$ and $\boldsymbol{c}_1^* \in \mathbb{C}(n, a, b, P)$ then $i < j$.*

Proof: We consider the following two cases.

- **Case 1:** If $i < j$ then the two received vectors are

$$\boldsymbol{c}_1 = (c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_j, c_{j+1}, \ldots, c_n),$$
$$\boldsymbol{c}_2 = (c_1, \ldots, c_{i-1}, c_i, \ldots, c_{j-1}, c_{j+1}, \ldots, c_n).$$

  Since $\boldsymbol{c}_1 \neq \boldsymbol{c}_2$, the two indices j_1 and j_2 always exist and $j_1 \leqslant j_2$. Without loss of generality, we assume that $j_1 = i$ and $j_2 = j - 1$, that is $c_i \neq c_{i+1}$ and $c_j \neq c_{j-1}$. In this case, we obtain $\boldsymbol{c}_1^* = \boldsymbol{c}_2^* = \boldsymbol{c} \in \mathbb{C}(n, a, b, P)$.
- **Case 2:** If $i > j$ then the two received vectors are

$$\boldsymbol{c}_1 = (c_1, \ldots, c_{j-1}, c_j, \ldots, c_{i-1}, c_{i+1}, \ldots, c_n),$$
$$\boldsymbol{c}_2 = (c_1, \ldots, c_{j-1}, c_{j+1}, \ldots, c_i, c_{i+1}, \ldots, c_n).$$

  We also assume that $j_1 = j$ and $j_2 = i - 1$, that is, $c_j \neq c_{j+1}$ and $c_i \neq c_{i-1}$. In this case, the two obtained vectors are

$$\boldsymbol{c}_1^* = (c_1, \ldots, c_{j-1}, c_{j+1}, c_j, \ldots, c_{i-2}, c_{i-1}, c_{i+1}, \ldots, c_n),$$
$$\boldsymbol{c}_2^* = (c_1, \ldots, c_{j-1}, c_{j+1}, c_{j+2}, \ldots, c_i, c_{i-1}, c_{i+1}, \ldots, c_n).$$

  We now consider both possibilities: $\boldsymbol{c}_1^* = \boldsymbol{c}_2^*$ and $\boldsymbol{c}_1^* \neq \boldsymbol{c}_2^*$.
  - **Case 2.1. $\boldsymbol{c}_1^* \neq \boldsymbol{c}_2^*$:** We can easily distinguish this case from Case 1.
  - **Case 2.2. $\boldsymbol{c}_1^* = \boldsymbol{c}_2^*$:** The subvector $(c_j, \ldots, c_i)$ has period two since $\boldsymbol{c}_1^*[j+1, i-1] = (c_j, \ldots, c_{i-2}) = (c_{j+2}, \ldots, c_i) = \boldsymbol{c}_2^*[j+1, i-1]$. Hence, the length of the subvector satisfies $i - j - 1 \leqslant P$, since $\boldsymbol{c} \in \mathbb{C}(n, a, b, P) \subseteq \mathbb{C}_2(n, P)$. Furthermore, we note that $\boldsymbol{c}_1^* \neq \boldsymbol{c} \in \mathbb{C}(n, a, b, P) \subseteq SVT_{a,b}(n, P)$. Assume that $\boldsymbol{c}_1^* \in SVT_{a,b}(n, P)$. Note that given the knowledge of the location of the deleted bit to within P consecutive positions and the received word $\boldsymbol{c}_1$, a decoder of the code $SVT_{a,b}(n, P)$ can not distinguish between $\boldsymbol{c}$ and $\boldsymbol{c}_1^*$. That is, the code $SVT_{a,b}(n, P)$ can not correct a single deletion, which is a contradiction. Hence, $\boldsymbol{c}_1^* \notin SVT_{a,b}(n, P)$.

From all these three possible outcomes, we can conclude that the three statements in the claim hold. ■

In step 2, after determining whether $i < j$ or $i > j$, it is easy to recover the original vector. In case $i < j$, by inserting $\boldsymbol{c}_2[j_1]$ into the j_1-th position in $\boldsymbol{c}_1$ we obtain the codeword $\boldsymbol{c}_1^* = \boldsymbol{c}$. Similarly, in the case $i > j$, we can also can recover the codeword by inserting $\boldsymbol{c}_2[j_2]$ into the j_2-th position in $\boldsymbol{c}_1$. In conclusion, Theorem 37 is proven.

ACKNOWLEDGMENT

The authors thank the two anonymous reviewers and the Associate Editor Prof. Vladimir Sidorenko for their valuable comments and suggestions, which have contributed for the clarity of the paper and its presentation.

REFERENCES

[1] J. Brakensiek, V. Guruswami, and A. Zbarsky. (Jul. 2015). "Efficient low-redundancy codes for correcting multiple deletions." [Online]. Available: https://arxiv.org/abs/1507.06175
[2] Y. Cassuto and M. Blaum, "Codes for symbol-pair read channels," *IEEE Trans. Inf. Theory*, vol. 57, no. 12, pp. 8011–8020, Dec. 2011.
[3] Y. M. Chee, H. M. Kiah, A. Vardy, V. K. Vu, and E. Yaakobi, "Coding for racetrack memories," in *Proc. IEEE Int. Symp. Inf. Theory*, Aachen, Germany, Jun. 2017, pp. 619–623.
[4] Y. M. Chee, H. M. Kiah, and C. Wang, "Maximum distance separable symbol-pair codes," in *Proc. IEEE Int. Symp. Inf. Theory*, Cambridge, MA, USA, Jul. 2012, pp. 2886–2890.
[5] Y. M. Chee, L. Ji, H. M. Kiah, C. Wang, and J. Yin, "Maximum distance separable codes for symbol-pair read channels," *IEEE Trans. Inf. Theory*, vol. 59, no. 11, pp. 7259–7267, Nov. 2013.
[6] L. Dolecek and V. Anantharam, "Repetition error correcting sets: Explicit constructions and prefixing methods," *SIAM J. Discrete Math.*, vol. 23, no. 4, pp. 2120–2146, 2010.
[7] R. Gabrys and E. Yaakobi, "Sequence reconstruction over the deletion channel," in *Proc. IEEE Int. Symp. Inf. Theory*, Barcelona, Spain, Jul. 2016, pp. 1596–1600.
[8] A. S. J. Helberg and H. C. Ferreira, "On multiple insertion/deletion correcting codes," *IEEE Trans. Inf. Theory*, vol. 48, no. 1, pp. 305–308, Jan. 2002.
[9] K. A. S. Immink, *Codes for Mass Data Storage Systems*, 2nd ed. Eindhoven, The Netherlands: Shannon Foundation Publishers, 2004.
[10] S. Jain, F. Farnoud, M. Schwartz, and J. Bruck. (2016). "Duplication-correcting codes for data storage in the DNA of living organisms." [Online]. Available: https://arxiv.org/abs/1606.00397
[11] V. I. Levenshtein, "Binary codes capable of correcting insertions, deletions and reversals," *Sov. Phys. Dokl.*, vol. 10, no. 8, pp. 707–710, 1966.
[12] V. I. Levenshtein, "Efficient reconstruction of sequences," *IEEE Trans. Inf. Theory*, vol. 47, no. 1, pp. 2–22, Jan. 2001.
[13] V. I. Levenshtein, "Efficient reconstruction of sequences from their subsequences or supersequences," *J. Combinat. Theory A*, vol. 93, no. 2, pp. 310–332, 2001.
[14] V. I. Levenshtein, "On perfect codes in deletion and insertion metric," *Discrete Math. Appl.*, vol. 2, no. 3, pp. 241–258, 1992.
[15] M. Levy and E. Yaakobi, "Mutually uncorrelated codes for DNA storage," in *Proc. IEEE Int. Symp. Inf. Theory*, Aachen, Germany, Jun. 2017, pp. 3115–3119.
[16] Z. Liu and M. Mitzenmacher, "Codes for deletion and insertion channels with segmented errors," in *Proc. IEEE Int. Symp. Inf. Theory*, Nice, France, Jun. 2007, pp. 846–850.
[17] M. Mitzenmacher, "A survey of results for deletion channels and related synchronization channels," *Probab. Surv.*, vol. 6, pp. 1–33, 2009.
[18] H. Mahdavifar and A. Vardy, "Nearly optimal sticky-insertion correcting codes with efficient encoding and decoding," in *Proc. IEEE Int. Symp. Inf. Theory*, Aachen, Germany, Jun. 2017, pp. 2683–2687.
[19] S. S. P. Parkin, M. Hayashi, and L. Thomas, "Magnetic domain-wall racetrack memory," *Science*, vol. 320, no. 5873, pp. 190–194, 2008.

[20] F. Sala, R. Gabrys, C. Schoeny, and L. Dolecek, "Exact reconstruction from insertions in synchronization codes," *IEEE Trans. Inf. Theory*, vol. 63, no. 4, pp. 2428–2445, Apr. 2017.
[21] M. F. Schilling, "The longest run of heads," *College Math. J.*, vol. 21, no. 3, pp. 196–207, 1990.
[22] M. F. Schilling, "The surprising predictability of long runs," *Math. Mag.*, vol. 85, no. 2, pp. 141–149, 2012.
[23] C. Schoeny, A. Wachter-Zeh, R. Gabrys, and E. Yaakobi, "Codes for correcting a burst of deletions or insertions," *IEEE Trans. Inf. Theory*, vol. 63, no. 4, pp. 1971–1985, Apr. 2017.
[24] Z. Sun, W. Wu, and H. Li, "Cross-layer racetrack memory design for ultra high density and low power consumption," in *Proc. Design Autom. Conf. (DAC)*, May/Jun. 2013, pp. 1–6.
[25] R. R. Varshamov and G. M. Tenengolts, "Codes which correct single asymmetric errors," (in Russian), *Avtomatika Telemekhanika*, vol. 26, no. 2, pp. 288–292, 1965.
[26] R. Venkatesan, V. Kozhikkottu, C. Augustine, A. Raychowdhury, K. Roy, and A. Raghunathan, "TapeCache: A high density, energy efficient cache based on domain wall memory," in *Proc. ACM/IEEE Int. Symp. Low Power Electron. Design (ISLPED)*, New York, NY, USA, Jul./Aug. 2012, pp. 185–190,
[27] F. Wang, D. Aktas, and T. M. Duman, "On capacity and coding for segmented deletion channels," in *Proc. 49th Annu. Allerton Conf. Allerton House*, Champaign, IL, USA, Sep., 2011, pp. 1408–1413.
[28] F. Wang, T. M. Duman, and D. Aktas, "Capacity bounds and concatenated codes over segmented deletion channels," *IEEE Trans. Commun.*, vol. 61, no. 3, pp. 852–864, Mar. 2013.
[29] A. J. V. Wijngaarden and K. A. S. Immink, "Construction of maximum run-length limited codes using sequence replacement techniques," *IEEE J. Sel. Areas Commun.*, vol. 28, no. 2, pp. 200–207, Feb. 2010.
[30] D. A. Wolfram, "Solving generalized Fibonacci recurrences," *Fibonacci Quart.*, vol. 36, no. 2, pp. 129–145, 1998.
[31] E. Yaakobi, J. Bruck, and P. H. Siegel, "Constructions and decoding of cyclic codes over *b*-symbol read channels," *IEEE Trans. Inf. Theory*, vol. 62, no. 4, pp. 1541–1551, Apr. 2016.
[32] C. Zhang *et al.*, "Hi-fi playback: Tolerating position errors in shift operations of racetrack memory," in *Proc. ACM/IEEE 42nd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jul. 2015, pp. 694–706.

Yeow Meng Chee (SM'08) received the B.Math. degree in computer science and combinatorics and optimization and the M.Math. and Ph.D. degrees in computer science from the University of Waterloo, Waterloo, ON, Canada, in 1988, 1989, and 1996, respectively.

Currently, he is a Professor at the Division of Mathematical Sciences, School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore. Prior to this, he was Program Director of Interactive Digital Media R&D in the Media Development Authority of Singapore, Postdoctoral Fellow at the University of Waterloo and IBMs Zürich Research Laboratory, General Manager of the Singapore Computer Emergency Response Team, and Deputy Director of Strategic Programs at the Infocomm Development Authority, Singapore.

His research interest lies in the interplay between combinatorics and computer science/engineering, particularly combinatorial design theory, coding theory, extremal set systems, and electronic design automation.

Han Mao Kiah received his Ph.D. degree in mathematics from Nanyang Technological University (NTU), Singapore, in 2014. From 2014 to 2015 he was a Postdoctoral Research Associate at the Coordinated Science Laboratory, University of Illinois at Urbana–Champaign. Currently he is a Lecturer at the School of Physical and Mathematical Sciences, NTU, Singapore. His research interests include combinatorial design theory, coding theory, and enumerative combinatorics.

Alexander Vardy (S'88–M'91–SM'94–F'99) was born in Moscow, U.S.S.R., in 1963. He earned his B.Sc. (*summa cum laude*) from the Technion, Israel, in 1985, and Ph.D. from the Tel-Aviv University, Israel, in 1991. During 1985–1990 he was with the Israeli Air Force, where he worked on electronic counter measures systems and algorithms. During the years 1992 and 1993 he was a Visiting Scientist at the IBM Almaden Research Center, in San Jose, CA. From 1993 to 1998, he was with the University of Illinois at Urbana-Champaign, first as an Assistant Professor then as an Associate Professor. Since 1998, he has been with the University of California San Diego (UCSD), where he is the Jack Keil Wolf Endowed Chair Professor in the Department of Electrical and Computer Engineering and the Department of Computer Science. While on sabbatical from UCSD, he has held long-term visiting appointments with CNRS, France, the EPFL, Switzerland, the Technion, Israel, and Nanyang Technological University, Singapore.

His research interests include error-correcting codes, algebraic and iterative decoding algorithms, lattices and sphere packings, coding for storage systems, cryptography and computational complexity theory, as well as fun math problems.

He received an IBM Invention Achievement Award in 1993, and NSF Research Initiation and CAREER awards in 1994 and 1995. In 1996, he was appointed Fellow in the Center for Advanced Study at the University of Illinois, and received the Xerox Award for faculty research. In the same year, he became a Fellow of the David and Lucile Packard Foundation. He received the IEEE Information Theory Society Paper Award (jointly with Ralf Koetter) for the year 2004. In 2005, he received the Fulbright Senior Scholar Fellowship, and the Best Paper Award at the IEEE Symposium on Foundations of Computer Science (FOCS). In 2017, his work on polar codes was recognized by the the IEEE Communications & Information Theory Societies Joint Paper Award.

During 1995–1998, he was an Associate Editor for Coding Theory and during 1998–2001, he was the Editor-in-Chief of the IEEE TRANSACTIONS ON INFORMATION THEORY. From 2003 to 2009, he was an Editor for the *SIAM Journal on Discrete Mathematics*. He is currently serving on the Executive Editorial Board for the IEEE TRANSACTIONS ON INFORMATION THEORY. He has been a member of the Board of Governors of the IEEE Information Theory Society during 1998–2006, and again during 2011–2017. He is a Fellow of the IEEE and the ACM.

Van Khu Vu received his B.Sc. degree in mathematics from Vietnam National University, Hanoi, in 2010. Currently he is pursuing the Ph.D. degree in mathematics from School of Physical and Mathematical Sciences at Nanyang Technological University, Singapore.

His primary research interests lie in the areas of algorithms, combinatorics and coding theory.

Eitan Yaakobi (S'07–M'12–SM'17) is an Assistant Professor at the Computer Science Department at the Technion Ð Israel Institute of Technology. He received the B.A. degrees in computer science and mathematics, and the M.Sc. degree in computer science from the Technion—Israel Institute of Technology, Haifa, Israel, in 2005 and 2007, respectively, and the Ph.D. degree in electrical engineering from the University of California, San Diego, in 2011. Between 2011-2013, he was a postdoctoral researcher in the department of Electrical Engineering at the California Institute of Technology. His research interests include information and coding theory with applications to non-volatile memories, associative memories, data storage and retrieval, and voting theory. He received the Marconi Society Young Scholar in 2009 and the Intel Ph.D. Fellowship in 2010–2011.