Codes Correcting Position Errors in Racetrack Memories

Yeow Meng Chee[*], **Han Mao Kiah**[*], **Alexander Vardy**[†*], **Van Khu Vu**[*], and **Eitan Yaakobi**[‡]

[*] School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore

[†] Department of Electrical and Computer Engineering, University of California San Diego, La Jolla, CA 92093, USA

[‡] Department of Computer Science, Technion — Israel Institute of Technology, Haifa, 32000 Israel

Emails:{ymchee,hmkiah,vankhu001}@ntu.edu.sg.edu, avardy@ucsd.edu, yaakobi@cs.technion.ac.il

***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 work we continue our recent study and 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. We will show how to take advantage of this special feature of racetrack memories in order to construct codes correcting deletions and sticky insertions.

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 [13].

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 [9].

There are several approaches to enhance the shift operation in order to reduce its time and energy consumption [10], [12]. However these mechanisms suffer from degraded reliability and cannot ensure that domains are perfectly shifted to they are aligned with the head. These errors, called *position errors*, can be modeled as deletions and sticky insertions [13], 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 with 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* [3] or *sticky insertions* in a *sticky channel* [3], [7], [8].

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], [4]–[6]. 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.

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], [5]. 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,

978-1-5090-3097-2/17/$31.00 ©2017 IEEE

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 review previous constructions we presented in [2]. In Section IV, we study codes correcting sticky insertions and in Section V, we present our main result in the paper of codes correcting both deletions and sticky insertions.

II. Preliminaries and Model Definitions

Let $\mathbb{F}_2$ denote the binary finite field and $\mathbb{F}_2^* = \bigcup_{n=0}^{\infty} \mathbb{F}_2^n$ is the set of all words over $\mathbb{F}_2$. For a positive integer n, the set $\{1, 2, \ldots, n\}$ is denoted by $[n]$. A subvector of a word $\boldsymbol{u}$ is a vector $\boldsymbol{u}[i_1, i_2] = (u_{i_1}, u_{i_1+1}, \ldots, u_{i_2}) \in \mathbb{F}_2^*$ 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 $\boldsymbol{u}$.

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.

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* d *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 1. 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:

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

□

As illustrated in Example 1, 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.

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*.

III. Previous Results

In our previous work [2], several code constructions were given which correct a single or multiple deletions. We briefly review these constructions here as they will be used for the codes we study in this paper.

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, $\mathbb{C}_1(n, 1, t) = \{\boldsymbol{c} \in \mathbb{F}_2^n \mid L(\boldsymbol{c}, 1) \leqslant t\}$.*

Construction 2. *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, $\mathbb{C}_2(n, b, t) = \{\boldsymbol{c} \in \mathbb{F}_2^n \mid L(\boldsymbol{c}, b) \leqslant t\}$.*

Construction 3. *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,*

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

The following results for these constructions were proved in [2].

Theorem 4.

1) *The code $\mathbb{C}_1(n,1,t)$ is a two-head single-deletion-correcting code when the heads are positioned t locations apart.*
2) *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.*
3) *The code $\mathbb{C}_3(n,\leqslant b,t)$ can correct up to b consecutive deletions using two heads at distance t.*
4) *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 $t=2(t_1-1)$.*

IV. 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 2. 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 [3], [8]. 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 5. *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. In particular, 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.*

V. 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 5, 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 6. *The code $\mathbb{C}_1(n,1,t)$ is a two-head single-position-error-correcting code if the distance between two heads is at least t.*

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

Theorem 7. *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 $\boldsymbol{c}_1 = \boldsymbol{c}$. We omit the proof due to the lack of space.

Claim 8. *$\boldsymbol{c} = \boldsymbol{c}_1$ 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.*

If one of the two conditions in Claim 8 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 8 $\boldsymbol{c}_1 \neq \boldsymbol{c}_2$ 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 9. *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:

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

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

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

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 (1), 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

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

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:

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

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 $\boldsymbol{c}$. Due to the lack of space and since this part repeats the ideas for the deletion case we omit this part of the proof.

Now we are ready to prove the correctness of the two decision rules. 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 (2), 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, it is possible to verify 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 (2) 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 9, 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 5.

The case of stick 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 4. ■

We observe that the proof of Theorem 7 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 3. 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:

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

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

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

According to the cardinality results for the code $\mathbb{C}_3(n, \leqslant 2, t_1)$ in [2], we conclude with the following corollary.

Corollary 10. *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* $t = 3(\lceil \log(n) \rceil + 2) - 2$.

REFERENCES

[1] J. Brakensiek, V. Guruswami, and A. Zbarsky, "Efficient low-redundancy codes for correcting multiple deletions," arxiv:1507.06175v1, Jul. 2015.

[2] Y.M. Chee, H.M. Kiah, A. Vardy, V.K. Vu, and E. Yaakobi, "Coding for racetrack memories," *Proc. IEEE Int. Symp. Inf. Theory*, Aachen, Germany, Jun. 2017.

[3] L. Dolecek and V. Anantharam, "Repetition error correcting sets: Explicit constructions and prefixing methods," *SIAM Journal on Discrete Mathematics*, vol. 23, no. 4, pp. 2120–2146, Jan. 2010.

[4] A.S.J. Helberg and Hendrik C. Ferreira, "On multiple insertion/deletion correcting codes," *IEEE Trans. Inf. Theory*, vol. 48, no. 1, pp. 305–308, Jan. 2002.

[5] V.I. Levenshtein, "Binary codes capable of correcting insertions, deletions and reversals", *Dokl. Akad. Nauk SSSR*, vol. 163, no. 4, pp. 845–848, 1965. English translation: *Sov. Phys. Dokl.*, vol. 10, no.8, pp. 707–710. 1966.

[6] V.I. Levenshtein, " On perfect codes in deletion and insertion metric," *Discr. Math.*, vol. 3, no. 1, pp. 3–20, 1991. English translation: *Discr. Math. Appl.*, vol. 2, no.3, pp. 241–258, 1992.

[7] M. Mitzenmacher, "A survey of results for deletion channels and related synchronization channels," *Probability Surveys*, vol. 6, pp. 1–33, 2009.

[8] H. Mahdavifar and A. Vardy, "Nearly optimal sticky-insertion correcting codes with efficient encoding and decoding," *Proc. IEEE Int. Symp. Inf. Theory*, Aachen, Germany, Jun. 2017.

[9] S.S. Parkin, M. Hayashi, and L. Thomas, "Magnetic domain-wall racetrack memory," *Science*, vol. 320, no. 5873, pp. 190–194, 2008.

[10] Z. Sun, W. Wu, and H. Li, "Cross-layer racetrack memory design for ultra high density and low power consumption," *Design Automation Conference (DAC)*, pp. 1–6, May 2013.

[11] R. R. Varshamov and G. M. Tenengolts, "Codes which correct single asymmetric errors (in Russian)," *Automatika i Telemkhanika*, vol. 161, no. 3, pp. 288–292, 1965.

[12] 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," *Proc. of the 2012 ACM/IEEE Int. Symp. on Low Power Electronics and Design (ISLPED)*, New York, NY, pp. 185–190, 2012.

[13] C. Zhang, G. Sun, X. Zhang, W. Zhang, W. Zhao, T. Wang, Y. Liang, Y. Liu, Y. Wang, and J. Shu, "Hi-fi playback: Tolerating position errors in shift operations of racetrack memory," *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pp. 694–706, Ju. 2015.