Reconstruction from Deletions in Racetrack Memories

Yeow Meng Chee*, **Ryan Gabrys**§, **Alexander Vardy**†*, **Van Khu Vu***, and **Eitan Yaakobi**‡

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

§ Spawar Systems Center, San Diego San Diego, CA, 92115

† 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,vankhu001`}`@ntu.edu.sg`, `ryan.gabrys@navy.mil`, `avardy@ucsd.edu`, `yaakobi@cs.technion.ac.il`

***Abstract*—In this work, we study a special case of the reconstruction problem in order to combat position errors in racetrack memories. In these memories, the information is stored in *magnetic cells* that can be sensed by shifting them under *read heads*. However, since this shifting operation is not error free, recent work has been dedicated towards correcting these so-called *position errors*, which manifest themselves as deletions and sticky insertions. A deletion is the event where the cells are over-shifted, and a sticky insertion occurs when the cells are not shifted.**

We first present a code construction that uses two heads to correct two deletions with at most $\log_2(\log_2 n) + 4$ redundant bits. This result improves upon a recent one that requires roughly $\log_2 n$ redundant bits. We then extend this construction to correct d deletions using d heads with at most $\log_2(\log_2 n) + c$ redundant bits. Lastly, we extend our results and derive codes for the classical reconstruction problem by Levenshtein over the insertion/deletion channel.

I. Introduction

Racetrack memory is an emerging non-volatile memory technology which has attracted significant attention in recent years due to its promising ultra-high storage density and low power consumption [13], [16]. The basic information storage element in a racetrack memory is called a *domain*, also known as a *cell*. The magnetization direction of each domain is programmed to store binary information. The reading mechanism is operated by a *read port*, called a *head*. Since all heads are fixed, each domain is shifted to its closest head to be read, by a *shift operation*. Since the shifting operation is not always accurate, it introduces errors, called *position errors*. The two kinds of position errors, caused by over-shifting and under-shifting the cells, can be modeled as deletions and sticky insertions, respectively [3], [18].

One natural approach to combat position errors in racetrack memories is to use classical insertion/deletion-correcting codes. These codes are required in a model that each cell is read by only a single head [17]. Although there are several known *deletion-correcting codes* [1], [7], [9] and *sticky-insertion-correcting codes* [5], [8], [12], only a little is known on codes correcting a combination of deletions and sticky insertions. Furtheremore, these codes may require a large number of redundant bits. For example, even for the easiest case of correcting a single sticky insertion, such a code requires at least $\log_2(n) - 1$ bits of redundancy, where n is the code length [5].

Another approach to tackle this problem is to leverage the special feature of racetrack memories, where multiple heads are used to read the information. Under this model, each cell of the stored word is read by multiple heads and a decoder receives multiple erroneous versions of the stored word. In fact, this model falls under the general framework of the reconstruction problem by Levenshtein [10], [11]. When over-shifts occur, that is, bits of the stored word are deleted in the outputs from all heads, Levenshtein presented efficient reconstruction algorithms to recover the stored word for the uncoded case, that is for all possible words [10], [11]. Recently, extensions for codes correcting a single deletion were studied in [6] and for insertions in [14]. Unfortunately, these algorithms require a large number of distinct noisy versions of the stored word.

The main difference between the model studied here and the one by Levenshtein is that for multiple heads in a racetrack memory, the position errors are correlated and depend upon the locations and distance between the different heads. For example, if two heads are t positions apart and the i-th bit is deleted in the first head then the $(i + t)$-th bit is deleted in the second head. Leveraging this special feature, several coding schemes were recently presented in [2], [3]. These schemes used the positions of the heads together with a coding constraint that uses at most a single bit of redundancy in order to guarantee that the heads' outputs are all different, for the case of a single over-shift. Moreover, efficient reconstruction algorithms to recover the stored word using a small number of heads were presented. Before we give an overview of these results, we introduce some necessary notations and terminology.

A. Notations

Let $\mathbb{F}_2$ denote the binary finite field. A *binary word* of length n over the alphabet $\mathbb{F}_2$ is a vector $\boldsymbol{u} \in \mathbb{F}_2^n$. For each word $\boldsymbol{u} = (u_1, \ldots, u_n) \in \mathbb{F}_2^n$, a subvector of the word $\boldsymbol{u}$ is a vector $\boldsymbol{u}[i_1, i_2] = (u_{i_1}, u_{i_1+1}, \ldots, u_{i_2}) \in \mathbb{F}_2^{i_2 - i_1 + 1}$, where $1 \leqslant i_1 \leqslant i_2 \leqslant n$. In case $i_1 = i_2 = i$, we denote the subvector $\boldsymbol{u}[i, i]$ by $\boldsymbol{u}[i]$ to specify the i-th bit u_i of the vector $\boldsymbol{u}$. For two vectors $\boldsymbol{u} = (u_1, \ldots, u_n) \in \mathbb{F}_2^n$ and $\boldsymbol{v} = (v_1, \ldots, v_m) \in \mathbb{F}_2^m$, the concatenation of $\boldsymbol{u}$ and $\boldsymbol{v}$ is the vector $(u_1, \ldots, u_n, v_1, \ldots, v_m) \in \mathbb{F}_2^{n+m}$, which is denoted by $\boldsymbol{u} \circ \boldsymbol{v}$.

A length-m vector $\boldsymbol{v} = (v_1, \ldots, v_m) \in \mathbb{F}_2^m$ is said to have *period* ℓ if $v_i = v_{i+\ell}$ for all $1 \leqslant i \leqslant m - \ell$. 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. Hence, $L(\boldsymbol{u}, 1)$ is the length of the longest run in $\boldsymbol{u}$.

A binary code of length n is a subset $\mathbb{C} \subseteq \mathbb{F}_2^n$. 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

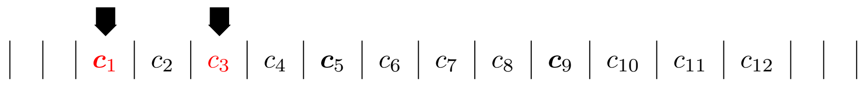

Fig. 1: Racetrack memory with twelve data domains and two heads

$|\mathbb{C}|$ is the size of the code $\mathbb{C}$ and the $\log$ here and in the rest of the paper is taken according to base 2.

For a length-n word $\boldsymbol{u} \in \mathbb{F}_2^n$ and $i \in [n]$, let $\boldsymbol{u}(\delta_i)$ be the vector obtained by a deletion of the i-th bit in the vector $\boldsymbol{u}$. 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.

A racetrack memory comprises n cells, each of which can store a single bit, and d heads, which are fixed. For example, in Fig. 1, a racetrack memory contains twelve data cells and two heads which are two positions apart. Let $\boldsymbol{c} = (c_1, c_2, \ldots, c_n) \in \mathbb{F}_2^n$ be a word stored in a racetrack memory where the i-th cell stores the bit c_i. We assume that there are two heads of fixed distance t and a shift operation is required for the heads to read all bits. Under this setup, if the first head reads the i-th bit c_i then the second head, which is t positions away, reads the $(i+t)$-th bit c_{i+t} and the output in each head is the stored word $\boldsymbol{c}$ if there is no error. However, a shift operation might not work perfectly. Let us consider an event where an over-shift occurs and every head skips the following bit. For example, when an over-shift occurs at the i-th position and the bit c_i is not read in the first head, then the bit c_{i+t} is not read in the second head. In this case, the output in the first head is $\boldsymbol{c}(\delta_i) = (c_1, \ldots, c_{i-1}, c_{i+1}, \ldots, c_n)$ and the output in the second head is $\boldsymbol{c}(\delta_{i+t}) = (c_1, \ldots, c_{i+t-1}, c_{i+t+1}, \ldots, c_n)$.

Example 1. Let $\boldsymbol{u} = (0,0,1,1,0,1,0,1,1,0,0,1) \in \mathbb{F}_2^{12}$ be the word stored in the memory, and assume that there are two heads which are positioned $t = 2$ positions apart. Assume that a deletion occurs at position 3 in the first head, then a deletion also occurs at position 5 in the second head. Hence, the outputs from the two heads are:

$$\textbf{Head 1: } \boldsymbol{u}(\delta_3) = (0,0,1,0,1,0,1,1,0,0,1),$$
$$\textbf{Head 2: } \boldsymbol{u}(\delta_5) = (0,0,1,1,1,0,1,1,0,0,1).$$

□

Similarly to over-shifts, when an under-shift occurs, each head reads the same bit again. In this case, if an under-shift occurs at position i in the first head, then the outputs in the two heads are:

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

In a racetrack memory, shift errors are also known as *position errors*. Our objective is to reconstruct the stored codeword from the erroneous outputs from all heads or, in other word, to correct positions errors. If there are many distinct outputs, we can use some known reconstruction algorithms to recover the stored word. However, in racetrack memory, the number of heads is limited and different heads may have the same output if the distance between their locations t is small and the stored word has a long run. In the worst case, where there is a single head or all heads have the same output, the only approach to correct the position errors is by using a code which is capable of correcting deletions and sticky insertions. This approach requires a large number of redundancy bits. Given h heads, our goal is to design codes correcting at most d deletions using h heads. Such a code is called an *h-head d-deletion-correcting code*. Similarly, we also define *h-head d-sticky-insertion-correcting code*. In the following, we seek to design codes with the minimum number of redundant bits.

In this paper, we study h-head d-deletion-correcting codes, while focusing on the special case of $h = 2$. The extended results on two-head d-deletion-correcting codes and h-head d-sticky-insertion-correcting codes will be presented in the extended version of this paper. An extension of our work with connection to the reconstruction problem by Levenshtein is discussed in Section III.

B. Previous Results

The problem of correcting position errors in a racetrack memory was recently studied in [3] and codes correcting position errors were presented. In particular, two-head single-deletion-correcting codes and $(d+1)$-head d-deletion-correcting codes were constructed as follows.

Construction 1. *[3] 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, $\mathbb{C}_1(n,1,t) = \{\boldsymbol{c} \in \mathbb{F}_2^n \mid L(\boldsymbol{c},1) \leqslant t\}$. Then, $\mathbb{C}_1(n,1,t)$ is a two-head single-deletion-correcting code, when the distance between the two heads is at least t.*

Construction 2. *[3] Given n, d, t, let*

$$\mathbb{C}_2(n, \leqslant d, t) = \{\boldsymbol{c} \in \Sigma_2^n \mid L(\boldsymbol{c},\ell) \leqslant t, \text{ for all } \ell \leqslant d\}.$$

Then, $\mathbb{C}_2(n, \leqslant d, t)$ is a $(d+1)$-head d-deletion-correcting code, when the distance between every two consecutive heads is at least $T(d) = t\left(\binom{d}{2}+1\right) + \frac{7d-d^3}{6}$.

Note that $\mathbb{C}_2(n, \leqslant 1, t) = \mathbb{C}_1(n,1,t)$. The codes in Constructions 1 and 2 can be seen as special families of constrained codes. In particular, the code $\mathbb{C}_1(n,1,t)$ is equivalent to a (d,k) run-length limited (RLL) constrained codes for $d = 0$ and $k = t-1$. While it is known that for every fixed t the rate of these codes is strictly less than 1, according to the following proposition, the redundancy of these codes will actually be very small when t is a function of the length n.

Proposition 3. *[3] For all n, d, t,*

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

As an immediate result from Proposition 3, the redundancy of the code $\mathbb{C}_2(n, \leqslant d, t)$ for $t = \lceil \log(n) \rceil + d + 1$ is at most a single bit. Furthermore, h-head d-deletion-correcting codes have been investigated in general. The following is the result in the case $h = d$.

Theorem 4. *[3] For all $d \geqslant 2$, there exists a d-head d-deletion-correcting code with most $\lceil \log(n+1) \rceil + 1$ redundancy bits, when the distance between every two consecutive heads is at least $\left(\binom{d}{2} + 1\right) \lceil \log(n) \rceil + \frac{d^3+5d+3}{3}$.*

Our first goal in this paper is to improve the redundancy in Theorem 4. Then, we show how to use our code for the general reconstruction problem over the deletion channel. Some of the proofs in the paper are omitted due to the lack of space.

C. Main Results

Our first contribution is a family of two-head double-deletion-correcting codes, which will be presented in Construction 9. We analyze the size of this code in Proposition 11 in order to obtain the following result, which improves upon the result in Theorem 4 for two deletions with respect to the number of redundancy bits.

Theorem 5. *There exists a two-head double-deletion-correcting code with redundancy of at most $\log \log n + 4$ bits when the distance between the two heads is at least $2(\lceil \log(n) \rceil + 2)$.*

A decoding algorithm of the above code can be derived from the proof of Lemma 10. Next, we leverage this result to construct d-head d-deletion-correcting codes for $d > 2$.

Theorem 6. *There exists a d-head d-deletion-correcting code with at most $\log \log n + c$ redundancy bits, where $c = \log(d(d-1) + 2)$, when the distance between every two consecutive heads is at least*

$$\left(\binom{d}{2} + 1\right) \lceil \log n \rceil + \frac{d^3 + 5d + 3}{3}.$$

Finally, we extend our result to the general reconstruction problem by Levenshtein [10], [11].

Theorem 7. *There exists a code with at most $\log \log n + 4$ redundancy bits that can reconstruct each codeword in this code using two distinct erroneous versions produced by a single deletion channel.*

Lastly, we note that our techniques can be extended also for codes correcting sticky insertions and this part will be presented in the extended version of this paper.

II. Code Constructions

The main goal in this section is to construct a two-head double-deletion-correcting code, which improves upon the result in Theorem 4. Our construction uses the family of *shifted Varshamov-Tenengolts* (*SVT*) codes, which were first presented in [15] for codes correcting bursts of deletions.

Construction 8. *[15] Given positive integers n, P and integers $0 \leqslant a < P, b \in \{0, 1\}$, let the shifted Varshamov-Tenengolts (SVT) code be:*

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

It was shown in [15] that the SVT code $SVT_{a,b}(n, P)$ can correct a single deletion if the position of the deletion is known to within at most P consecutive positions [15]. In this section, SVT codes are used as one of the component codes in a two-head double-deletion-correcting code.

Construction 9. *Let n, P, d, t_1 be positive integers and $0 \leqslant a < P, 0 \leqslant b < 2$. Let*

$$\mathbb{C}_3(n, a, b, P, d, t_1) = \mathbb{C}_2(n, \leqslant d, t_1) \cap SVT_{a,b}(n, P),$$

where the code $\mathbb{C}_2(n, \leqslant d, t_1)$, $SVT_{a,b}(n, P)$ is constructed as in Construction 2, Construction 8, respectively.

The following proposition shows the correctness of the above construction.

Lemma 10. *The code $\mathbb{C}_3(n, a, b, P, 2, t_1)$ is a two-head double-deletion-correcting code when the distance between the two heads is at least $t = 2(t_1 - 1)$ and $P = 3t_1 - 5$.*

Proof: Let $\boldsymbol{c} = (c_1, \ldots, c_n) \in \mathbb{C}_3(n, a, b, P, 2, t_1)$ be the stored codeword and $t = 2(t_1 - 1)$ be the distance between the two heads. Assume that the two deletions occurred in the first head are in positions i_1, i_2, where $i_1 < i_2$. Hence the two deletions in the second head are in positions $i_1 + t$ and $i_2 + t$. The outputs from the two heads are:

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

We prove that it is possible to correct these two deletions by explicitly showing the decoding procedure. This will be done in the following three steps:

1) **Step 1:** Correct the first deletion in the first head by reconstructing (as shown in [3]).
2) **Step 2:** Estimate the location of the second deletion in the first head to within some P consecutive positions.
3) **Step 3:** Use the decoder of the SVT code to recover the stored codeword.

In Step 1, we simply repeat the arguments in [3] to show how to correct the first deletion in the first head. We repeat these steps for the completeness of the proof and since we need this analysis for Steps 2 and 3.

First, 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 to the contrary that there is equality. Then, we distinguish between the following two cases:

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

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

are identical, and thus the subvector $(c_{i_1},\ldots,,c_{i_1+t_1})$ forms a run of length t_1+1, in contradiction to the construction of the code $\mathbb{C}_2(n,\leqslant 2,t_1)$.

- **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 following two subvectors

$$\boldsymbol{c}_1[i_1,i_2+t_1-3]=(c_{i_1+1},\ldots,c_{i_2-1},c_{i_2+1},\ldots,c_{i_2+t_1-1}),$$
$$\boldsymbol{c}_2[i_1,i_2+t_1-3]=(c_{i_1},\ldots,c_{i_2-2},c_{i_2-1},\ldots,c_{i_2+t_1-3})$$

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 $\mathbb{C}_2(n,\leqslant 2,t_1)$.

Thus,

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

Moreover,

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

Hence, there exists j_1 which is the leftmost index such that $\boldsymbol{c}(\delta_{i_1},\delta_{i_2})$ and $\boldsymbol{c}(\delta_{i_1+t},\delta_{i_2+t})$ differ 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 $n-j_1-1$ bits in $\boldsymbol{c}(\delta_{i_1},\delta_{i_2})$, that is,

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

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 furthermore $\boldsymbol{c}(\delta_{i_1+t},\delta_{i_2+t})[1,j_1]=(c_1,\ldots,c_{j_1})$. 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, $(c_{i_2-1},c_{i_2},\ldots,c_{j_1},c_{j_1+1})$ 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 $0\leqslant k_1\leqslant t_1-2$. In other words, by concatenating $\boldsymbol{c}(\delta_{i_1+t},\delta_{i_2+t})[1,j_1]$ and $\boldsymbol{c}(\delta_{i_1},\delta_{i_2})[j_1,n-2]$, we obtain the vector $\boldsymbol{c}(\delta_{i_2+k_1})$ which is also obtained from the vector $\boldsymbol{c}$ by deleting the bit in location i_2+k_1, where $0\leqslant k_1\leqslant t_1-2$.

Next, in Step 2, we estimate the location i_2 of the second deletion in the first head and thus estimate the value of i_2+k_1. Let $\boldsymbol{r}$ be the reverse vector of $\boldsymbol{c}$, that is, $\boldsymbol{r}=\overleftarrow{\boldsymbol{c}}$ or $r_i=c_{n+1-i}$ for all $1\leqslant i\leqslant n$. Let $\boldsymbol{r}_1,\boldsymbol{r}_2$ be the reverse vector of $\boldsymbol{c}_1,\boldsymbol{c}_2$, respectively. Then, $\boldsymbol{r}_1=\boldsymbol{r}(\delta_{n+1-i_2},\delta_{n+1-i_1})$ and $\boldsymbol{r}_2=\boldsymbol{r}(\delta_{n+1-i_2-t},\delta_{n+1-i_1-t})$. Applying the same arguments as in Step 1, there exists j_2 which is the leftmost index that $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ differ and $n+1-i_2-t\leqslant j_2\leqslant n-i_2$. Hence, $n+1-j_2-t\leqslant i_2\leqslant n-j_2$ and thus $n+1-j_2-t\leqslant i_2+k_1\leqslant n-j_2+t_1-2$ since $0\leqslant k_1\leqslant t_1-2$. Therefore, we can estimate the location of the deletion i_2+k_1 in a segment of length $3t_1-5$ from $n+1-j_2-t$ to $n-j_2+t_1-2$.

In the last step, we simply choose $P=3t_1-5$ and use the decoder of the code $SVT_{a,b}(n,P)$ with the input $\boldsymbol{c}(\delta_{i_2+k_1})$ obtained in Step 1, and the range of positions for its deletion in order to recover the stored codeword $\boldsymbol{c}$. ■

Next we compute the size and redundancy of the code $\mathbb{C}_3(n,a,b,P,2,t_1)$, and thus obtain the result in Theorem 5.

Proposition 11. *For all n,t_1 and $P=(3t_1-5)$, there exist $0\leqslant a<P,b\in\{0,1\}$ such that the size of the code constructed in Construction 9 satisfies*

$$|\mathbb{C}_3(n,a,b,P,2,t_1)|\geqslant\frac{2^n}{2(3t_1-5)}\left(1-n\cdot\left(\frac{1}{2}\right)^{t_1-2}\right).\quad(1)$$

In particular, for $t_1=\lceil\log(n)\rceil+3$ the redundancy of the code $\mathbb{C}_3(n,a,b,P,2,t_1)$ is at most $\log\log n+4$ bits.

Proof: Note that $\mathbb{C}_2(n,\leqslant 2,t_1)\geqslant 2^n\cdot\left(1-n\cdot\left(\frac{1}{2}\right)^{t_1-2}\right)$ from Proposition 3. We consider all $2P$ cosets of the SVT code $SVT_{a,b}(n,P)$ which form a partition of the space $\mathbb{F}_2^n$. Then, according to the pigeonhole principle with $P=(3t_1-5)$ there exist $0\leqslant a<P,b\in\{0,1\}$ such that

$$\mathbb{C}_3(n,a,b,P,2,t_1)\geqslant\frac{2^n}{2(3t_1-5)}\left(1-n\cdot\left(\frac{1}{2}\right)^{t_1-2}\right).$$

Therefore, for $t_1=\lceil\log(n)\rceil+3$, the redundancy is at most $\log(4P)\leqslant\log\log n+4$ bits. ■

From the proof of Lemma 10, it is possible to use two outputs to correct the first deletion in the first output if there are only two deletions in each output. The result can be generalized to the case there are d deletions in each output as follows.

Lemma 12. *[Lemma 17, [4]] 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 positions, such that for all $1\leqslant\ell\leqslant d$,*

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

Assume $\boldsymbol{c}\in\mathbb{C}_2(n,\leqslant d,t_1)$ 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 $2\leqslant\ell\leqslant d$,

$$i_{1,\ell}\leqslant i'_{1,\ell}\leqslant i_{1,\ell}+(d-1)t_1-d(d-1)/2+1.\quad(3)$$

In case there are d heads and $\Delta_h=\{\delta_{i_{h,1}},\ldots,\delta_{i_{h,d}}\}$ is the positions set of d deletions in the h-th head for $1\leqslant h\leqslant d$, we consider $d-1$ pairs of outputs $\{c(\Delta_i),c(\Delta_{i+1})\}$ for $1\leqslant i\leqslant d-1$. From Lemma 12, we can correct the first deletion in the first $d-1$ heads to obtain $d-1$ vectors. Repeating this procedure $d-2$ times, we can correct $d-2$ deletions and obtain two vectors with two deletions in each vector. Now, we use Lemma 10 to correct these two deletions to recover the original vector. We omit the details and state the following result instead.

Theorem 13. *The code $\mathbb{C}_3(n,a,b,P,d,t_1)$ is a d-head d-deletion-correcting code if the distance between any two heads is at least $t\geqslant T(d)$, where $P=T(d)+t_1-3$ and*

$$T(d)=t_1\left(\binom{d}{2}+1\right)+\frac{7d-d^3}{6}.$$

By choosing $t_1=\lceil\log n\rceil+d+1$, the code $\mathbb{C}_3(n,a,b,P,d,t_1)$ requires at most $2+\log P\leqslant\log\log n+c$ bits of redundancy, where $c=\log(d(d-1)+2)$ and n is large enough. Hence, we conclude that our code is a d-head d-deletion-correcting code with at most $\log\log n + \log(d(d-1)+2)$ bits of redundancy, when the distance between consecutive heads is at least

$$\left(\binom{d}{2}+1\right)\lceil \log n\rceil + \frac{d^3+5d+3}{3}.$$

Thus, we establish the result in Theorem 6.

III. Connections to the Reconstruction Problem

In this section, we investigate a problem of correcting shift-errors in racetrack memories. With the special feature of having multiple heads in racetrack memories, we obtain multiple outputs. However, each of them has multiple deletions due to shift errors in racetrack memories. Hence, this problem is closely related to the Levenshtein's reconstruction problem [10], [11]. Thus far, our main contribution is the construction of the code correcting two deletions using two erroneous outputs in racetrack memories. We note that in our model, the deletions in different outputs are correlated, depend on the distance between two heads. This fact does not hold in general case of Levenshtein's problem. Fortunately, our constructed code still works in general case of the classical reconstruction problem if there is only single deletion.

Theorem 14. *Given a word $\boldsymbol{c} \in \mathbb{C}_3(n,a,b,P,2,t_1)$, using only two distinct noisy versions (produced by a single deletion channel), we can reconstruct the word $\boldsymbol{c}$ exactly.*

Proof: Let $\boldsymbol{c} = (c_1,\ldots,c_n)$ be a stored codeword and $\boldsymbol{c}_1, \boldsymbol{c}_2$ be two received vectors. Let c_i (respectively c_j) be a deleted bit in $\boldsymbol{c}_1$ (respectively $\boldsymbol{c}_2$). That is, $\boldsymbol{c}_1 = (c_1,\ldots,c_{i-1},c_{i+1},\ldots,c_n)$ and $\boldsymbol{c}_2 = (c_1,\ldots,c_{j-1},c_{j+1},\ldots,c_n)$.

The theorem is proved by showing an explicit decoding procedure, which is described in Algorithm 1. It is possible to

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

```
Input: $\boldsymbol{c}_1,\boldsymbol{c}_2 \in \mathbb{F}_2^{n-1}$.
Output: $\boldsymbol{c} \in \mathbb{F}_2^n$.
  1. $j_1 \leftarrow$ leftmost index that $\boldsymbol{c}_1 \neq \boldsymbol{c}_2$;
  2. $j_2 \leftarrow$ rightmost index that $\boldsymbol{c}_1 \neq \boldsymbol{c}_2$;
  3. $\boldsymbol{c}_1^* \leftarrow \boldsymbol{c}_1[1,j_1-1] \circ \{\boldsymbol{c}_2[j_1]\} \circ \boldsymbol{c}_1[j_1,n-1]$;
  4. $\boldsymbol{c}_2^* \leftarrow \boldsymbol{c}_2[1,j_2-1] \circ \{\boldsymbol{c}_1[j_2]\} \circ \boldsymbol{c}_2[j_2,n-1]$;
  5.
  if $\boldsymbol{c}_1^* \neq \boldsymbol{c}_2^*$ then
      return $\boldsymbol{c} = \boldsymbol{c}_1[1,j_2-1] \circ \{\boldsymbol{c}_2[j_2]\} \circ \boldsymbol{c}_1[j_2,n-1]$;
  6.
  if $\boldsymbol{c}_1^* = \boldsymbol{c}_2^*$ and $\boldsymbol{c}_1^* \notin \mathbb{C}_3(n,a,b,P,2,t_1)$ then
      return $\boldsymbol{c} = \boldsymbol{c}_1[1,j_2-1] \circ \{\boldsymbol{c}_2[j_2]\} \circ \boldsymbol{c}_1[j_2,n-1]$;
  7.
  if $\boldsymbol{c}_1^* = \boldsymbol{c}_2^*$ and $\boldsymbol{c}_1^* \in \mathbb{C}_3(n,a,b,P,2,t_1)$ then
      return $\boldsymbol{c} = \boldsymbol{c}_1^*$;
```

show that Algorithm 1 can successfully reconstruct the stored word $\boldsymbol{c}$ with linear complexity. ■

From Proposition 11, we know that the redundancy of the code $\mathbb{C}_3(n,a,b,P,2,t_1)$ is at most $\log\log n + 4$ bits. Hence, we can obtain the result in Theorem 7. The details of this section can be found in our full paper [4].

IV. Conclusion

In this work, we studied coding techniques to combat shift-errors (position errors) in racetrack memories. These errors were modeled as deletions and sticky insertions. Leveraging the special feature of having multiple read ports in racetrack memories, constrained codes have been proposed to reconstruct the stored codeword from multiple erroneous outputs. In this work, we combine constrained codes with SVT codes to design a new code correcting these position errors with a fewer number redundancy bits, compared to previous known results. We also analyze the connection between this problem and the classical reconstruction problem by Levenshtein to show that our code is useful for the classical reconstruction problem. Our techniques can be generalized for d-head d-sticky-insertion-correcting codes.

V. Acknowledgement

The research of Y. M. Chee is partially funded by the Singapore Ministry of Education under grants MOE2017-T3-1-007 and MOE2015-T2-2-086. The research of Alexander Vardy was supported in part by the National Science Foundation under Grants CCF-1405119 and CCF-1719139. The research of E. Yaakobi is partially supported by the German Science Foundation (GIF) Grant I-1356-407.6/2016.

References

[1] J. Brakensiek, V. Guruswami, and S. Zbarsky, "Efficient low-redundancy codes for correcting multiple deletions," *IEEE Trans. Inform. Theory*, vol. 64, no. 5, pp. 3403–3410, May 2018.

[2] Y. M. Chee, H. M. Kiah, A. Vardy, V. K. Vu, and E. Yaakobi, "Codes correcting position errors in racetrack memories," in *Proc. IEEE Inform. Theory Workshop*, 2017, pp. 161–165.

[3] Y. M. Chee, H. M. Kiah, A. Vardy, V. K. Vu, and E. Yaakobi, "Coding for racetrack memories," in *Proc. IEEE Int. Symp. on Inform. Theory*, 2017, pp. 619–623.

[4] Y. M. Chee, H. M. Kiah, A. Vardy, V. K. Vu, and E. Yaakobi, "Coding for racetrack memories," to appear in *IEEE Trans. Inform. Theory*.

[5] 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.

[6] R. Gabrys and E. Yaakobi, "Sequence reconstruction over the deletion channel", *IEEE Trans. Inform. Theory*, vol. 64, no. 4, pp. 2924–2931, Apr. 2018.

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

[8] S. Jain, F. Farnoud, M. Schwartz, and J. Bruck, "Duplication-correcting codes for data storage in the DNA of living organisms", *IEEE Trans. Inform. Theory*, vol. 63, no. 8, pp. 4996–5010, Aug. 2017.

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

[10] V. I. Levenshtein, "Efficient reconstruction of sequences," *IEEE Trans. Inform. Theory*, vol. 47, no. 1, pp. 2–22, Jan. 2001.

[11] V. I. Levenshtein, "Efficient reconstruction of sequences from their subsequences or supersequences", *J. of Combin. Theory, Ser. A*, vol. 93, no. 2, pp. 310–332, 2001.

[12] H. Mahdavifar and A. Vardy, "Asymptotically optimal sticky-insertion correcting codes with efficient encoding and decoding," in *Proc. IEEE Int. Symp. on Inform. Theory*, 2017, pp. 2683–2687.

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

[14] F. Sala, R. Gabrys, C. Schoeny, and L. Dolecek, "Exact reconstruction from insertions in synchronization codes", *IEEE Trans. Inform. Theory*, vol. 63, no. 4, pp. 2428–2445, Apr. 2017.

[15] C. Schoeny, A. Wachter-Zeh, R. Gabrys, and E. Yaakobi, "Codes for correcting a burst of deletions or insertions," *IEEE Trans. Inform. Theory*, vol. 63, no. 4, pp. 1971–1985, Apr. 2016.

[16] 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.

[17] A. Vahid, G. Mappouras, D. J. Sorin, and R. Calderbank, "Correcting two deletions and insertions in racetrack memory", online in https://arxiv.org/abs/1701.06478.

[18] 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," *ACM/IEEE 42nd Annual Int. Symp. on Computer Architecture (ISCA)*, pp. 694–706, Jul. 2015.