Coding for 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 has a tape-like structure 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 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. 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.

I. Introduction

Racetrack memory, also known as *domain wall memory*, is an emerging non-volatile memory which is based on spintronics technology. It attracts significant attention due to its promising ultra-high storage density, even comparing to other spintronics memory technologies such as STT-RAM [16].

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 a 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 [13], [15]. 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 [16], 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* or *sticky insertions* in a *sticky channel* [2], [8].

In this work we study codes which correct position errors in racetrack memory. At first sight, this problem is not any different than the well-studied problem of designing codes correcting deletions and insertions [1], [5]. However, we take 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 reading 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* [6]. However, in our case, as opposed to the general one studied by Levenshtein, the position errors are correlated and depend on the locations and distance between the different heads.

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 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 cor-

978-1-5090-4096-4/17/$31.00 ©2017 IEEE

recting 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 where the length of the burst is either exactly b or at most b. 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, by requiring the distance between adjacent heads to be at least $d\lceil \log(n) \rceil + d(d+1)/2 + 1$. In the 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 remaining $d-(m-1)$ deletions. In this section, we also report on several more results we could not include due to the lack space. Some of the proofs are omitted for the same reason.

II. Preliminaries and Model Definitions

Let $\mathbb{F}_2$ denote the binary field. For a positive integer n, the set $\{1, 2, \ldots, n\}$ is denoted by $[n]$. 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 $\boldsymbol{u}[i_1, i_2] = (u_{i_1}, u_{i_1+1}, \ldots, u_{i_2}) \in \mathbb{F}_2^{i_2-i_1+1}$ 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 $\boldsymbol{u} = (u_1, \ldots, u_9) = (0, 0, 1, 1, 0, 1, 0, 1, 1) \in \mathbb{F}_2^9$. 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 $\boldsymbol{u}(\delta_{[3,4]}) = (0, 0, 0, 1, 1)$. □

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

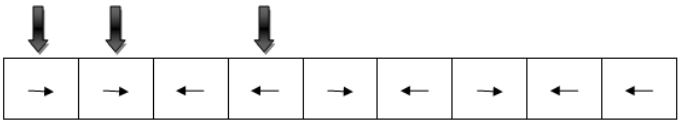

Fig. 1: Racetrack memory with multiple heads

which moves all the cells on each cycle one or more steps in the same direction [9]. 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.

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 there are three heads which are used to read the stored word $\boldsymbol{u}$, where the distance between the first two heads is t_1 and the distance between the last two heads is t_2. 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.

The goal in this paper is to design codes correcting 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 is also part of the code design. Since the area for shifting the cells is constrained, the heads should not be too far apart and the distance between adjacent heads should thus 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.

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 following theorem proves the correctness of this construction.

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 than 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 also provides a simple decoding algorithm for the code $\mathbb{C}_1(n,1,t)$. ∎

The next example demonstrates this code construction.

Example 3. 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 thus the stored codeword is decoded according to $\boldsymbol{c}=\boldsymbol{c}(\delta_6)[1,4]\circ\boldsymbol{c}(\delta_3)[4,8]=(0,0,1,1,0,1,0,1,1)$. □

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 [3]. While efficient encoding and decoding algorithms are known for codes which satisfy the $(0,t-1)$ RLL constraint for fixed value of t, the rates of these codes is strictly less than 1. Since we can achieve codes with rate approaching 1 by simply using a single head and a single-deletion-correcting code of redundancy at most $\log(n+1)$ [5], we are interested only in codes with rate approaching 1 and will optimize their redundancy. Thus, we follow a similar approach to the one taken in [12] for codes correcting a burst 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 [12], using the derivations from [10] and [11], that the redundancy of the code $\mathbb{C}_1(n,1,\lceil\log(n)\rceil+1)$ is approximately 0.36, and for $t=\lceil\log(n)\rceil+2$ efficient encoding and decoding algorithms were recently found for these codes using a single bit of redundancy [7]. 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, so a burst of deletions occurs in each head. 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 from 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.*

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. Let us start with the following definition.

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

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

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

- $\Phi:\mathbb{C}_2(n,b,t)\to\mathbb{F}_2^b\times\mathbb{R}(n-b,t-b)$, where $\Phi(\boldsymbol{u})=(\boldsymbol{u}[1,b],\boldsymbol{p}_b(\boldsymbol{u}))$.
- $\Psi:\mathbb{F}_2^b\times\mathbb{R}(n-b,t-b)\to\mathbb{C}_2(n,b,t)$, 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 [4], 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, $|\mathbb{C}_2(n, b, t)| = 2^b \cdot |\mathbb{R}(n - b, t - b)|$.*

The size of the code $\mathbb{R}(n, t)$ can be calculated using the results from Section III and by applying Lemma 8 for $b = 1$ to get that for all n and $t \leqslant n$, $|\mathbb{R}(n, t)| = |\mathbb{C}_1(n + 1, 1, t + 1)|/2$. We can now conclude with the following corollary.

Corollary 9. *For all n, b, t,*

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

According to Corollaries 3 and 9 we conclude the following.

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,*

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

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

In this case we will not able to provide an exact approximation for the redundancy of the code $\mathbb{C}_3(n, \leqslant b, t)$ as in previous cases. 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. in [12] when studying the redundancy of the so-called *universal RLL constraint*. This result is stated 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.

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 m heads. 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.

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, use the first two heads to correct the first deletion in the first head.
2) Then, 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.

In order to prove the first step, 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 in the contrary, then we distinguish between the following two cases:

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

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

  are identical, so 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 2: If $i_2 - i_1 \leqslant t_1$ then $i_1 + t = i_1 + 2t_1 - 2 \geqslant i_2 + t_1 - 2$. Therefore, the first $i_2 + t_1 - 3$ bits in the first two heads, which are subvectors

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

  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. It is possible to show that by concatenating 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 $\boldsymbol{c}(\delta_{i_1}, \delta_{i_2})$, the resulting vector is $\boldsymbol{c}(\delta_{i_2+k_1}) = \boldsymbol{c}(\delta_{i_1+t}, \delta_{i_2+t})[1, j_1] \circ \boldsymbol{c}(\delta_{i_1}, \delta_{i_2})[j_1, n - 2]$, where $0 \leqslant k_1 \leqslant t_1 - 2$. In a similar method, it is possible to obtain the vector $\boldsymbol{c}(\delta_{i_2+t+k_2})$, for $0 \leqslant k_2 \leqslant t_1 - 2$. Lastly, in the third

step we have two vectors $\boldsymbol{c}(\delta_{i_2+k_1})$ and $\boldsymbol{c}(\delta_{i_2+t+k_2})$, where $(i_2+t+k_2)-(i_2+k_1) \geqslant (i_2+t)-(i_2+t_1-2) = t_1$. Therefore, following the proof in Theorem 2, we can reconstruct the codeword $\boldsymbol{c}$, thereby correcting the two deletions. ■

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

Example 4. Let $n = 11, t_1 = 3, t = 4$ and the stored codeword is $\boldsymbol{c} = (0,0,1,1,0,1,1,0,1,1,1) \in \mathbb{C}_2(11, \leqslant 2, 3)$. Assume that the outputs from the three heads are:

$$\textbf{Head 1: } \boldsymbol{c}(\delta_1, \delta_3) = (0,1,0,1,1,0,1,1,1),$$
$$\textbf{Head 2: } \boldsymbol{c}(\delta_5, \delta_7) = (0,0,1,1,1,0,1,1,1),$$
$$\textbf{Head 3: } \boldsymbol{c}(\delta_9, \delta_{11}) = (0,0,1,1,0,1,1,0,1).$$

By comparing the outputs from the first two heads, we see that $j_1 = 2$ is the leftmost index that $\boldsymbol{c}(\delta_1, \delta_3)$ and $\boldsymbol{c}(\delta_5, \delta_7)$ differ. Hence, we can obtain the vector

$$\boldsymbol{c}(\delta_3) = \boldsymbol{c}(\delta_5, \delta_7)[1,2] \circ \boldsymbol{c}(\delta_1, \delta_3)[2,9] = (0,0,1,0,1,1,0,1,1,1).$$

Similarly, we find the leftmost index that $\boldsymbol{c}(\delta_5, \delta_7)$ and $\boldsymbol{c}(\delta_9, \delta_{11})$ differ which is $j_2 = 5$ and obtain the vector

$$\boldsymbol{c}(\delta_7) = \boldsymbol{c}(\delta_9, \delta_{11})[1,5] \circ \boldsymbol{c}(\delta_5, \delta_7)[5,9] = (0,0,1,1,0,1,0,1,1,1).$$

Now, we can recover the original codeword by finding $j_3 = 4$ as the leftmost index that $\boldsymbol{c}(\delta_3)$ and $\boldsymbol{c}(\delta_7)$ differ and recover the stored codeword $\boldsymbol{c}$ to be

$$\boldsymbol{c} = \boldsymbol{c}(\delta_7)[1,4] \circ \boldsymbol{c}(\delta_3)[4,10] = (0,0,1,1,0,1,1,0,1,1,1). \quad \square$$

Based on the cardinality result on the code $\mathbb{C}_3(n, \leqslant 2, t_1)$ from Section IV we conclude with the following corollary.

Corollary 15. *There exists a three-head double-deletion-correcting code with at most a single bit of redundancy when the distance between adjacent heads is at least $t = 2(\lceil \log(n) \rceil + 2)$.*

The idea in the proof of Theorem 14 was to use every two pairs of adjacent heads in order to correct the first deletion in the first head in each pair. It turns that this basic procedure is all we need in order to generalize the construction to m-head d-deletion-correcting codes. Due to the lack of space we only state here the results of these constructions.

Theorem 16. *Let $\mathbb{C}$ be a $(d-m+1)$-deletion-correcting code, where $m \leqslant d+1$. 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 $t \geqslant t_1 + \sum_{k=1}^{d} \big((k-1)t_1 - k(k-1)/2 + 1\big)$. 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 $\lceil \log(n+1) \rceil + 1$.*

Lastly, we report on our results for the other cases solved in this work, which we could not include their details.

Theorem 17.

1) *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.*
2) *The code $\mathbb{C}_1(n,1,t)$ is a two-head single-position-error-correcting code when the distance between two heads is at least t.*
3) *The code $\mathbb{C}_3(n, \leqslant 2, t_1)$ is a three-head two-position-error-correcting code when the distance between adjacent heads is at least $t = 3t_1 - 2$.*

VI. Acknowledgement

The research of Y. M. Chee was supported in part by the Singapore Ministry of Education under Research Grant MOE2015-T2-2-086. The research of H. M. Kiah was supported in part by the Singapore Ministry of Education under Research Grants MOE2015-T2-2-086 and MOE2016-T1-001-156. The research of Alexander Vardy was supported in part by the National Science Foundation under Grant CCF-1405119. The research of Eitan Yaakobi was supported in part by the Israel Science Foundation (ISF) Grant 1624/14.

References

[1] J. Brakensiek, V. Guruswami, and A. Zbarsky, "Efficient low-redundancy codes for correcting multiple deletions," arxiv:1507.06175v1, Jul. 2015.
[2] 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.
[3] K.A.S.Immink, "Codes for mass data storage systems", *2nd ed.Eindhoven, The Netherlands: Shannon Foundation Publishers,* 2004.
[4] S. Jain, F. Farnoud, M. Schwartz, and J. Bruck, "Duplication-correcting codes for data storage in the DNA of living organisms", abs/1606.00397, 2016.
[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 trans.: *Sov. Phys. Dokl.*, vol. 10, no. 8, pp. 707–710. 1966.
[6] V.I. Levenshtein, "Efficient reconstruction of sequences," *IEEE Trans. Inf. Theory*, vol. 47, no. 1, pp. 2–22, Jan. 2001.
[7] M. Levy and E. Yaakobi, "Mutually uncorrelated codes for DNA storage," in *Proc. IEEE Int. Symp. on Inform. Theory*, Aachen, Germany, Jun. 2017.
[8] H. Mahdavifar and A. Vardy, "Nearly optimal sticky-insertion correcting codes with efficient encoding and decoding," in *Proc. IEEE Int. Symp. on Inform. 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] M. F. Schilling, "The longest run of heads," *College Math. J*, vol. 21, no. 3, pp. 196–207, 1990.
[11] M. F. Schilling, "The surprising predictability of long runs," *Mathematics Magazine*, vol. 85, no. 2, pp. 141–149, 2012.
[12] C. Schoeny, A. Wachter-Zeh, R. Gabrys, and E. Yaakobi, "Codes for correcting a burst of deletions or insertions," to appear *IEEE Trans. Inf. Theory*, 2017.
[13] Z. Sun, W. Wu, and H. Li, "Cross-layer racetrack memory design for ultra high density and low power consumption," in *Design Automation Conference (DAC)*, pp. 1–6, May 2013.
[14] 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.
[15] 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. of the 2012 ACM/IEEE Int. Symp. on Low Power Electronics and Design (ISLPED)*, New York, NY, pp. 185–190, 2012.
[16] 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 Int. Symp. on Computer Architecture (ISCA)*, pp. 694–706, Ju. 2015.