Universal Rewriting in Constrained Memories

Anxiao (Andrew) Jiang
Computer Science Department
Texas A&M University
College Station, TX 77843, U.S.A.
ajiang@cs.tamu.edu

Michael Langberg
Computer Science Division
Open University of Israel
Raanana 43107, Israel
mikel@openu.ac.il

Moshe Schwartz
Electrical and Computer Eng.
Ben-Gurion University
Beer Sheva 84105, Israel
schwartz@ee.bgu.ac.il

Jehoshua Bruck
EE & CNS Dept.
Caltech
Pasadena, CA 91125, U.S.A.
bruck@paradise.caltech.edu

***Abstract*—A constrained memory is a storage device whose elements change their states under some constraints. A typical example is flash memories, in which cell levels are easy to increase but hard to decrease. In a general rewriting model, the stored data changes with some pattern determined by the application. In a constrained memory, an appropriate representation is needed for the stored data to enable efficient rewriting.**

In this paper, we define the general rewriting problem using a graph model. This model generalizes many known rewriting models such as floating codes, WOM codes, buffer codes, etc. We present a novel rewriting scheme for the flash-memory model and prove it is asymptotically optimal in a wide range of scenarios.

We further study randomization and probability distributions to data rewriting and study the expected performance. We present a randomized code for *all* rewriting sequences and a deterministic code for rewriting following *any* i.i.d. distribution. Both codes are shown to be optimal asymptotically.

I. Introduction

Many storage media have constraints on their state transitions. A typical example is flash memory, the most widely used type of non-volatile electronic memory [4]. A multi-level flash memory cell has q levels: $0, 1, \ldots, q-1$. It is easy to increase a cell level but very costly to decrease it because to decrease the level of a *single* cell, a whole block of $\sim 10^5$ cells needs to be erased and reprogrammed [4]. Other storage media, including magnetic recording, optical recording and some new memory materials, have constraints on state transitions as well.

A storage medium needs to change its state when the stored data changes its value. Depending on the applications, the data often changes under some constrained patterns. For example, the data may change altogether or have its individual components rewritten asynchronously [13]. In another example, when the data represents an information stream, it changes in a sliding window fashion [2]. Thus, an appropriate representation is needed for the data to enable efficient rewriting.

We present the general model of constrained memories and rewriting using graph notation.

Definition 1. *(*Constrained Memory*) A constrained memory is represented by a directed graph $\mathcal{M} = (V_{\mathcal{M}}, E_{\mathcal{M}})$. The vertices $V_{\mathcal{M}}$ represent all the memory states. There is a directed edge (u, v) from $u \in V_{\mathcal{M}}$ to $v \in V_{\mathcal{M}}$ iff the memory can change from state u to state v without going through any other intermediate states. $\mathcal{M}$ is called the* memory graph.

Example 2. *(*Flash Memory Model*) For a flash memory with n cells of q levels each, the memory graph $\mathcal{M}$ has q^n vertices. Every vertex can be represented by a vector $(c_1, c_2, \ldots, c_n)$, where $c_i \in \{0, 1, \ldots, q-1\}$ is the i-th cell level, for $i = 1, \ldots, n$. There is a directed edge from $(c_1, c_2, \ldots, c_n)$ to $(c'_1, c'_2, \ldots, c'_n)$ iff there exists exactly one index $i \in \{1, \ldots, n\}$ such that $c'_i = c_i + 1$ while $c'_j = c_j$ for $j = 1, \ldots, i-1, i+1, \ldots, n$.*[1]

Definition 3. *(*Generalized Rewriting*) The stored data is represented by a directed graph $\mathcal{D} = (V_{\mathcal{D}}, E_{\mathcal{D}})$. The vertices $V_{\mathcal{D}}$ represent all the values that the data can take. There is a directed edge (u, v) from $u \in V_{\mathcal{D}}$ to $v \in V_{\mathcal{D}}$, $v \neq u$, iff a rewriting operation may change the stored data from value u to value v. The graph $\mathcal{D}$ is called the* data graph *and the number of its vertices, corresponding to the input-alphabet size, is denoted by $L = |V_{\mathcal{D}}|$. Throughout the paper we assume all data graphs to be strongly connected.*

Example 4. *(*Rewriting in Floating Codes [13]*) The data consists of k variables $v_1, \ldots, v_k$, each of which takes its value from the alphabet $\{0, 1, \ldots, \ell-1\}$. Every rewrite changes the value of one variable. Hence, the data graph $\mathcal{D}$ has $L = \ell^k$ vertices. Every vertex can be represented by a vector $(v_1, v_2, \ldots, v_k)$, where $v_i \in \{0, 1, \ldots, \ell-1\}$ is the i-th variable's value, for $i = 1, \ldots, k$. There is a directed edge from $(v_1, v_2, \ldots, v_k)$ to $(v'_1, v'_2, \ldots, v'_k)$ iff there exists a single index $i \in \{1, \ldots, k\}$, such that $v'_i \neq v_i$, while $v'_j = v_j$ for all $j \neq i$. The floating code model reduces to the write-once memory (WOM) code model [18] when $k = 1$. It can be seen that the data graph $\mathcal{D}$ is a generalized hypercube of k dimensions. When $k = 1$, it is a complete graph of order ℓ.*

Example 5. *(*Rewriting in Buffer Codes [2]*) A buffer code stores the last r values in a data stream similar to a FIFO queue. The stored data consists of k variables $v_1, \ldots, v_k$, each of which takes its value from the alphabet $\{0, 1, \ldots, \ell-1\}$. With every rewrite, v_i takes v_{i+1}'s old value for all $i = 1, \ldots, k-1$, and v_k takes the most recent value in the streaming data. Thus, the data graph $\mathcal{D}$ has $L = \ell^k$ vertices. Each vertex can be represented by a vector $(v_1, v_2, \ldots, v_k)$, where $v_i \in \{0, 1, \ldots, \ell-1\}$ is the i-th variable's value, for $i = 1, \ldots, k$. There is a directed edge from $(v_1, v_2, \ldots, v_k)$ to $(v'_1, v'_2, \ldots, v'_k)$ iff $v'_i = v_{i+1}$ for $i = 1, \ldots, k-1$. It can be seen that the data graph $\mathcal{D}$ is a de Bruijn graph.*

Definition 6. *(*Code for Rewriting*) A code for rewriting has a* decoding function F_d *and an* update function F_u*. The decoding function $F_d : V_{\mathcal{M}} \to V_{\mathcal{D}}$ maps a memory state $s \in V_{\mathcal{M}}$ to the stored data $F_d(s) \in V_{\mathcal{D}}$. The update function (which*

[1] This is a special case of the generalized write-once memory model in [6].

represents a rewriting operation), $F_u : V_{\mathcal{M}} \times V_{\mathcal{D}} \to V_{\mathcal{M}}$, maps the current memory state $s \in V_{\mathcal{M}}$ and the new data $v \in V_{\mathcal{D}}$ to a memory state $F_u(s,v) \in V_{\mathcal{M}}$ such that $F_d(F_u(s,v)) = v$. Clearly, there should be a directed path from s to $F_u(s,v)$ in the memory graph $\mathcal{M}$.

We note that if $F_d(s) = v$ we may set $F_u(s,v) = s$, which corresponds to a case in which we do not need to change the stored data. Throughout the paper we do not consider such a case as a rewrite operation.

A sequence of rewrites is a sequence $(v_0, v_1, v_2 \dots)$ such that the i-th rewrite changes the stored data from v_{i-1} to v_i. Given a storage code for rewriting $\mathcal{C}$, we denote by $t(\mathcal{C})$ the number of rewrites that $\mathcal{C}$ guarantees to support for all rewrite sequences. Thus, $t(\mathcal{C})$ is a worst-case performance measure of the code. The code $\mathcal{C}$ is said to be *optimal* if $t(\mathcal{C})$ is maximized. On the other side, if a probabilistic model for rewriting or randomization for code construction is used, the expected rewriting performance can be defined accordingly.

In this paper, we study general rewriting for the flash-memory model.[2] We present a novel code construction, the *trajectory code*, based on tracing the changes of data in the data graph $\mathcal{D}$. The code is asymptotically optimal (up to constant factors) for a very wide range of scenarios. It includes floating codes, WOM codes, and buffer codes as special cases, and is a substantial improvement compared to known results.

We further study randomization and probability distributions to data rewriting and study the expected performance. A code is called *strongly robust* if its asymptotic expected performance is optimal for *all* rewriting sequences. It is called *weakly robust* if the asymptotic expected performance is optimal for rewriting following *any* i.i.d. distribution. We present a randomized construction for strongly robust code and a deterministic construction for weakly robust code.

Both our codes for general rewriting and our robust codes are optimal up to constant factors (factors independent of the problem parameters). Namely, for a constant $r \leqslant 1$, we present codes $\mathcal{C}$ for which $t(\mathcal{C})$ is at least r times that of the optimal code. We would like to note that for our robust codes the constant involved is arbitrarily close to 1.

The rest of the paper is organized as follows. In Section II, related results are reviewed and compared to the results in this paper. In Section III, a new code construction, the *trajectory code*, is presented and its optimality is analyzed. In Section IV, robust codes are presented. In Section V we briefly discuss the results of the paper.

II. Overview of Related Results

There has been a history of distinguished theoretical study on constrained memories. It includes the original work by Kuznetsov and Tsybakov on coding for defective memories [15]. Further developments on defective memories include [10], [12]. The write once memory (WOM) [18], write unidirectional memory (WUM) [19]–[21], and write efficient memory [1], [9] are also special instances of constrained memories. Among them, WOM is the most related to the flash memory model studied in this paper.

Write once memory (WOM) was studied by Rivest and Shamir in their original work [18]. In a WOM, a cell's state can change from 0 to 1 but not from 1 to 0. This model was later generalized with more cell states in [6], [8]. The objective of WOM codes is to maximize the number of times that the stored data can be rewritten. A number of very interesting WOM code constructions have been presented over the years, including the tabular codes, linear codes, etc. in [18], the linear codes in [6], the codes constructed using projective geometries [17], and the coset coding in [5]. Profound results on the capacity of WOM have been presented in [8], [11], [18], [22]. Furthermore, error-correcting WOM codes have been studied in [25]. In all the above works, the rewriting model assumes no constraints on the data, namely, the data graph $\mathcal{D}$ is a complete graph.

With the increasing importance of flash memories, the flash memory model was proposed and studied recently in [2], [13]. The rewriting schemes include floating codes [13], [14] and buffer codes [2]. Both types of codes use the joint coding of multiple variables for better rewriting capability. Their data graphs $\mathcal{D}$ are generalized hypercubes and de Bruijn graphs, respectively. Multiple floating codes have been presented, including the code constructions in [13], [14], the flash codes in [16], [24], and the constructions based on Gray codes in [7]. The floating codes in [7] were optimized for the expected rewriting performance.

Compared to existing codes, the codes in this paper are not only for a more general rewriting model, but also provide asymptotically-optimal performance for a wider range of cases. This can be seen clearly from Table I, where the asymptotically-optimal codes are summarized.

III. Trajectory Code

We use the flash memory model of Example 2 and the generalized rewriting model of Definition 3 in the rest of this paper. We first present a novel code construction, the *trajectory code*, then show its asymptotically-optimal performance.

A. Trajectory Code Outline

Let $n_0, n_1, n_2, \dots, n_d$ be $d+1$ positive integers and let $n = \sum_{i=0}^{d} n_i$, where n denotes the number of flash cells, each of q levels. We partition the n cells into $d+1$ groups, each with $n_0, n_1, \dots, n_d$ cells, respectively. We call them *registers* $S_0, S_1, \dots, S_d$, respectively.

Our encoding uses the following basic scheme: we start by using register S_0, called the *anchor*, to record the value of the initial data $v_0 \in V_{\mathcal{D}}$. For the next d rewrite operations we use a differential scheme: denote by $v_1, \dots, v_d \in V_{\mathcal{D}}$ the next d values of the rewritten data. In the i-th rewrite, $1 \leqslant i \leqslant d$, we store in register S_i the identity of the edge $(v_{i-1}, v_i) \in E_{\mathcal{D}}$. We do not require a unique label for all edges globally, but rather require that *locally*, for each vertex in $V_{\mathcal{D}}$, its out-going

[2]The codes here are more suitable for NOR flash memories, which allow random access of cells. NAND flash memories have much more restricted access modes for cell pages, which limit usable coding schemes on rewriting.

TABLE I
A SUMMARY OF THE CODES FOR REWRITING WITH ASYMPTOTICALLY OPTIMAL PERFORMANCE (UP TO CONSTANT FACTORS). HERE n, k, ℓ, L ARE AS DEFINED IN EXAMPLES 2, 4, 5.

TYPE	ASYMPTOTIC OPTIMALITY	REF.
WOM code ($\mathcal{D}$ is a complete graph)	$t(\mathcal{C})$ is asymptotically optimal	[18]
WOM code ($\mathcal{D}$ is a complete graph)	$t(\mathcal{C})$ is asymptotically optimal when $\ell = \Theta(1)$	[6]
floating code ($\mathcal{D}$ is a hypercube)	$t(\mathcal{C})$ is asymptotically optimal when $k = \Theta(1)$ and $\ell = \Theta(1)$	[13] [14]
floating code ($\mathcal{D}$ is a hypercube)	$t(\mathcal{C})$ is asymptotically optimal when $n = \Omega(k \log k)$ and $\ell = \Theta(1)$	[13] [14]
floating code ($\mathcal{D}$ is a hypercube)	$t(\mathcal{C})$ is asymptotically optimal when $n = \Omega(k^2)$ and $\ell = \Theta(1)$	[24]
buffer code ($\mathcal{D}$ is a de Bruijn graph)	$t(\mathcal{C})$ is asymptotically optimal when $n = \Omega(k)$ and $\ell = \Theta(1)$	[2] [23]
floating code ($\mathcal{D}$ is a hypercube)	weakly robust codes when $k = \Theta(1)$ and $\ell = 2$	[7]
WOM code ($\mathcal{D}$ is a complete graph)	$t(\mathcal{C})$ is asymptotically optimal when $n = \Omega(\log^2 \ell)$	this paper
more general coding ($\mathcal{D}$ has maximum out-degree Δ. For floating codes, $\Delta = k(\ell - 1)$.)	$t(\mathcal{C})$ is asymptotically optimal when $n = \Omega(L)$, or when $n = \Omega(\log^2 L)$ and $\Delta = O(\frac{n \log n}{\log L})$. When $n = \Omega(\log^2 L)$, $t(\mathcal{C})$ is asymptotically optimal in the worst case sense (worst case over all data graphs $\mathcal{D}$).	this paper
robust coding	Strongly robust codes when $L^2 \log L = o(qn)$. Weakly robust codes when $L = \Theta(1)$.	this paper

edges have unique labels from $\{1, \ldots, \Delta\}$, where Δ denotes the maximal out-degree in the data graph $\mathcal{D}$.

Intuitively, the first d rewrite operations are achieved by encoding the *trajectory* taken by the input sequence starting with the anchor data. After d such rewrites, we repeat the process by rewriting the next input from $V_{\mathcal{D}}$ in the anchor S_0, and then continuing with d edge labels in $S_1, \ldots, S_d$.

Let us assume a sequence of s rewrites have been stored thus far. To decode the last stored value all we need to know is $s \mod (d+1)$. This is easily achieved by using $\lceil t/q \rceil$ more cells (not specified in the previous $d+1$ registers), where t is the total number of rewrite operations we would like to guarantee. For these $\lceil t/q \rceil$ cells we employ a simple encoding scheme: in every rewrite operation we arbitrarily choose one of those cells and raise its level by one. Thus, the total level in these cells equals s.

The decoding process takes the value of the anchor S_0 and then follows $(s-1) \mod (d+1)$ edges which are read consecutively from $S_1, S_2, \ldots$. Notice that this scheme is appealing in cases where the maximum out-degree of $\mathcal{D}$ is significantly lower than the state space $V_{\mathcal{D}}$.

Note that each register S_i, for $i = 0, \ldots, d$, can be seen as a *smaller rewriting code* whose data graph is a *complete graph* of either L vertices (for S_0) or Δ vertices (for $S_1, \ldots, S_d$). We let $d = 0$ if $\mathcal{D}$ is a complete graph, and describe how to set d when $\mathcal{D}$ is not a complete graph in section III-C. The encoding used by each register is described in the next section.

B. Analysis for a Complete Data Graph

In this section we present an efficiently encodable and decodable code that enables us to store and rewrite symbols from an input alphabet $V_{\mathcal{D}}$ of size $L \geqslant 2$, and where $\mathcal{D}$ is a complete graph. The information is stored in n flash cells of q levels each. (To use the code for register S_i with $i > 0$, we just need to replace L by Δ.)

We first state a scheme that allows approximately $nq/8$ rewrites in the case in which $2 \leqslant L \leqslant n$. We then extend it to hold for general L and n. We present the quality of our code constructions (namely the number of possible rewrites they perform) using the $\Theta(f)$ notation. Here, for functions f and g, we say that $g = \Theta(f)$ if g is asymptotically bounded both above and below by f up to a constant factor independent of the variables of f and g.

1) The Case $2 \leqslant L \leqslant n$: In this section we present a code for small values of L. The code we present is essentially the one presented in [18].

Construction 7. *Let $2 \leqslant L \leqslant n$. This construction is an efficiently encodable and decodable rewriting code $\mathcal{C}$ for a complete data graph $\mathcal{D}$ with L states, and flash memory with n cells with q states each.*

Let us first assume $n = L$. Denote the n cell levels by $\vec{c} = (c_0, c_1, \ldots, c_{L-1})$, where $c_i \in \{0, 1, \ldots, q-1\}$ is the level of the i-th cell for $i = 0, 1, \ldots, L-1$. Denote the alphabet of data by $V_{\mathcal{D}} = \{0, 1, \ldots, L-1\}$. We first use only cell levels 0 and 1, and the data stored in the cells is $\sum_{i=0}^{L-1} i c_i \pmod L$. With each rewrite, we increase the minimum number of cell levels from 0 to 1 so that the new cell state represents the new data. (Clearly, c_0 remains untouched as 0.) When the code can no longer support rewriting, we increase all cells (including c_0) from 0 to 1, and start using cell levels 1 and 2 to store data in the same way as above, except that the data stored in the cells uses the formula $\sum_{i=0}^{L-1} i(c_i - 1) \pmod L$. This process is repeated $q-1$ times in total. The general decoding function is therefore defined as

$$F_d(\vec{c}) = \sum_{i=0}^{L-1} i(c_i - c_0) \pmod L.$$

We now extend the above code to $n \geqslant L$ cells. We divide the n cells into $b = \lfloor n/L \rfloor$ groups of size L (some cells may remain unused). We first apply the code above to the first group of L cells, then to the second group, and so on.

Theorem 8. *Let $2 \leqslant L \leqslant n$. The code $\mathcal{C}$ in Construction 7 guarantees $t(\mathcal{C}) = n(q-1)/8 = \Theta(nq)$ rewrites.*

Proof: First assume $n = L$. When cell levels $j-1$ and j are used to store data (for $j = 1, \ldots, q-1$), by the analysis in [18], even if only one or two cells increase their levels with each rewrite, at least $(L+4)/4$ rewrites can be supported. So the L cells can support at least $\frac{(L+1)(q-1)}{4}$ rewrites. Now let $n \geqslant L$. When $b = \lfloor n/L \rfloor$, it is easy to see that $bL \geqslant n/2$.

The b groups of cells can guarantee $t(\mathcal{C}) = \frac{b(L+4)(q-1)}{4} \geqslant \frac{n(q-1)}{8} = \Theta(nq)$ rewrites. ∎

2) The Case of Large L: We now consider the typical setting in which L is larger than n. The rewriting code we present reduces the general case to that of the case $n = L$ studied above. We start by assuming that $n < L \leqslant 2^{\sqrt{n}}$. We will address the general case at the end of this section.

Let b be the smallest positive integer value that satisfies $\lfloor n/b \rfloor^b \geqslant L$.

Claim 9. *For $16 \leqslant n \leqslant L \leqslant 2^{\sqrt{n}}$ it holds that*

$$b \leqslant \frac{2\log L}{\log n}.$$

Proof: We first note that for all $1 \leqslant x \leqslant \frac{\sqrt{n}}{2}$, we have $\lfloor n/x \rfloor^x \geqslant n^{x/2}$. Since $16 \leqslant n \leqslant L \leqslant 2^{\lfloor\sqrt{n}\rfloor}$, it is easy to verify that indeed

$$\frac{2\log L}{\log n} \leqslant \frac{\sqrt{n}}{2}.$$

Therefore,

$$\left\lfloor \frac{n\log n}{2\log L} \right\rfloor^{\frac{2\log L}{\log n}} \geqslant n^{\frac{\log L}{\log n}} = L,$$

which implies the upper bound. ∎

Construction 10. *Let $n < L \leqslant 2^{\sqrt{n}}$. This construction is an efficiently encodable and decodable rewriting code $\mathcal{C}$ for a complete data graph $\mathcal{D}$ with L states, and flash memory with n cells with q states each.*

For $i = 1, 2, \ldots, b$, let v_i be a symbol from an alphabet of size $\lfloor n/b \rfloor \geqslant L^{1/b}$. We may represent any symbol $v \in V_{\mathcal{D}}$ as a vector of symbols $(v_1, v_2, \ldots, v_b)$. Partition the n flash cells into b groups, each with $\lfloor n/b \rfloor$ cells (some cells may remain unused). Encoding the symbol v into n cells is equivalent to the encoding of each v_i into the corresponding group of $\lfloor n/b \rfloor$ cells. As the alphabet size of each v_i equals the number of cells it is to be encoded into, we can use Construction 7 to store v_i.

Theorem 11. *Let $16 \leqslant n \leqslant L \leqslant 2^{\sqrt{n}}$. The code $\mathcal{C}$ in Construction 10 guarantees*

$$t(\mathcal{C}) = \frac{n(q-1)\log n}{16\log L} = \Theta\left(\frac{nq\log n}{\log L}\right)$$

rewrites.

Proof: Using Construction 10, the number of rewrites possible is bounded by the number of rewrites possible for each of the b cell groups. By Theorem 8 and Claim 9, this is at least

$$\left\lfloor \frac{n}{b} \right\rfloor \cdot \frac{q-1}{8} \geqslant \left(\frac{n\log n}{2\log L} - 1\right)\frac{q-1}{8} = \Theta\left(\frac{nq\log n}{\log L}\right).$$

∎

C. Analysis for a Bounded Out-Degree Data Graph

We now return to the outline of the trajectory code from Section III-A and apply it in full detail using the codes from Section III-B2 to the case of data graphs $\mathcal{D}$ with bounded out-degree Δ. We refer to such graphs as Δ-restricted. To simplify our presentation, in the theorems below we will again use the $\Theta(f)$ notation freely, however, as opposed to the previous section we will no longer state or make an attempt to optimize the constants involved in our calculations. We assume that $n \leqslant L \leqslant 2^{\sqrt{n}}$. Notice that for $L \leqslant n$, Construction 7 can be used to obtain optimal codes (up to constant factors).

Using the notation of Section III-A, to realize the trajectory code we need to specify the sizes n_i and the value of d. We consider two cases: the case in which Δ is *small* compared to n, and the case in which Δ is *large*.

Construction 12. *Let $\Delta \leqslant \left\lfloor \frac{n\log n}{2\log L} \right\rfloor$. We build an efficiently encodable and decodable rewriting code $\mathcal{C}$ for any Δ-restricted data graph $\mathcal{D}$ with L vertices and n flash cells of q levels as follows. For the trajectory code, let $d = \lfloor \log L / \log n \rfloor = \Theta(\log L / \log n)$. Set the size of the $d+1$ registers to $n_0 = \lfloor n/2 \rfloor$ and $n_i = \lfloor n/(2d) \rfloor \geqslant \Delta$ for $i = 1, \ldots d$. (We obviously have $\sum n_i \leqslant n$.)*

The update and decoding functions of the trajectory code $\mathcal{C}$ are defined as follows: Consider using the encoding scheme specified in Construction 10 for the encoding of symbols from $V_{\mathcal{D}}$ in the n_0 flash cells of S_0 corresponding to the anchor, and using the scheme specified in Construction 7 for the encoding of one of $\{1, \ldots, \Delta\}$ in the flash cells of S_i ($i = 1, \ldots, d$). Notice that the latter is possible as $n_i \geqslant \Delta$ for $i = 1, \ldots d$.

Theorem 13. *Let $\Delta \leqslant \left\lfloor \frac{n\log n}{2\log L} \right\rfloor$. The code $\mathcal{C}$ of Construction 12 guarantees $t(\mathcal{C}) = \Theta(nq)$ rewrites.*

Proof: By Theorems 11 and 8, the number of rewrites possible in S_0 is equal (up to constant factors) to that of S_i ($i \geqslant 1$):

$$\Theta\left(\frac{n_0 q\log n_0}{\log L}\right) = \Theta\left(\frac{nq\log n}{\log L}\right) = \Theta\left(\frac{nq}{d}\right) = \Theta\left(n_i q\right)$$

Thus the total number of rewrites in the scheme outlined in Section III-A is $d+1$ times the bound for each register S_i, and so $t(\mathcal{C}) = \Theta(nq)$. ∎

Construction 14. *Let $\left\lfloor \frac{n\log n}{2\log L} \right\rfloor \leqslant \Delta \leqslant L$. We build an efficiently encodable and decodable rewriting code $\mathcal{C}$ for any Δ-restricted data graph $\mathcal{D}$ with L vertices and n flash cells of q levels as follows. For the trajectory code, let $d = \lfloor \log L / \log \Delta \rfloor = \Theta(\log L / \log \Delta)$. Set the size of the registers to $n_0 = \lfloor n/2 \rfloor$ and $n_i = \lfloor n/(2d) \rfloor$ for $i = 1, \ldots d$.*

The update and decoding functions of the trajectory code $\mathcal{C}$ are defined as follows: Consider using the encoding scheme specified in Construction 10 for both the encoding of symbols from $V_{\mathcal{D}}$ in the n_0 flash cells of S_0 corresponding to the anchor, and the encoding of one of $\{1, \ldots, \Delta\}$ in the flash cells of S_i ($i = 1, \ldots, d$).

Theorem 15. *Let* $\left\lfloor \frac{n \log n}{2 \log L} \right\rfloor \leqslant \Delta \leqslant L$. *The code* $\mathcal{C}$ *of Construction* 14 *guarantees* $t(\mathcal{C}) = \Theta\left(\frac{nq \log n}{\log \Delta}\right)$ *rewrites.*

Proof: By Theorem 11, the number of rewrites possible in S_0 is:

$$\Theta\left(\frac{n_0 q \log n_0}{\log L}\right) = \Theta\left(\frac{nq \log n}{\log L}\right)$$

Similarly the number of rewrites possible in S_i ($i \geqslant 1$):

$$\Theta\left(\frac{n_i q \log n_i}{\log \Delta}\right) = \Theta\left(\frac{nq \log n}{d \log \Delta}\right) = \Theta\left(\frac{nq \log n}{\log L}\right).$$

Here we use the fact that as $d \leqslant \log L$ it holds that $d = o(n)$ and $\log n_i = \Theta(\log n - \log d) = \Theta(\log n)$. Notice that the two expressions above are equal. Thus, as in Theorem 13, we conclude that the total number of rewrites in the scheme outlined in Section III-A is $d+1$ times the bound for each register S_i, and so $t(\mathcal{C}) = \Theta\left(\frac{nq \log n}{\log \Delta}\right)$. ∎

D. Optimality of the Schemes

We describe upper bounds on the number of rewrites in general rewriting schemes to complement the lower bounds induced by our constructions.

Theorem 16. *Any rewriting code* $\mathcal{C}$ *that stores symbols from some data graph* $\mathcal{D}$ *in* n *flash cells of* q *levels supports at most* $t(\mathcal{C}) \leqslant n(q-1) = O(nq)$ *rewrites.*

Proof: The bound is trivial. In the best case, all cells are initialized at level 0, and every rewrite increases exactly one cell by exactly one level. Thus, the total number of rewrites is bounded by $n(q-1) = O(nq)$ as claimed. ∎

For large values of L, we can improve the upper bound. First, let r denote the largest integer such that $\binom{r+n-1}{r} < L-1$. We need the following technical claim.

Claim 17. *For all* $1 \leqslant n < L-1$, *it holds that*

$$r \geqslant \max\left\{ \left\lfloor \frac{\log(L-1)}{1+\log n} \right\rfloor, 1 \right\}.$$

Proof: First, it is easy to see that $r \geqslant 1$ since

$$\binom{1+n-1}{1} = n < L-1.$$

Next, when $\left\lfloor \frac{\log(L-1)}{1+\log n} \right\rfloor \geqslant 1$ we may use the well-known bound for all $v \geqslant u \geqslant 1$,

$$\binom{v}{u} < \left(\frac{ev}{u}\right)^u,$$

where e is the base of the natural logarithm. Thus,

$$\binom{\left\lfloor \frac{\log(L-1)}{1+\log n} \right\rfloor + n - 1}{\left\lfloor \frac{\log(L-1)}{1+\log n} \right\rfloor} < \left(\frac{e\left\lfloor \frac{\log(L-1)}{1+\log n} \right\rfloor + en - e}{\left\lfloor \frac{\log(L-1)}{1+\log n} \right\rfloor} \right)^{\left\lfloor \frac{\log(L-1)}{1+\log n} \right\rfloor}$$
$$\leqslant (en)^{\frac{\log(L-1)}{1+\log n}} = L-1,$$

which proves our claim. ∎

Theorem 18. *When* $n < L-1$, *any rewriting code* $\mathcal{C}$ *that stores symbols from some data graph* $\mathcal{D}$ *in* n *flash cells of* q *levels supports at most* $t(\mathcal{C}) = O\left(\frac{nq \log n}{\log L}\right)$ *rewrites.*

Proof: Let us examine some state s of the n flash cells, currently storing some value $v \in V_{\mathcal{D}}$, i.e., $F_d(s) = v$. Having no constraint on the input transition graph, the next symbol we want to store may be any of the $L-1$ symbols $v' \in V_{\mathcal{D}}$, $v' \neq v$.

If we allow ourselves r operations of increasing a single cell level of the n flash cells (perhaps, operating on the same cell more than once), we may reach $\binom{n+r-1}{r}$ distinct new states. However, by our choice $\binom{n+r-1}{r} < L-1$ and so we need at least $r+1$ such operations in the worst case. Since we have a total of n cells with q levels each, the number of rewrite operations is upper bounded by

$$t(\mathcal{C}) \leqslant \frac{n(q-1)}{r+1} \leqslant \frac{n(q-1)}{\left\lfloor \frac{\log(L-1)}{1+\log n} \right\rfloor + 1} = O\left(\frac{nq \log n}{\log L}\right).$$

∎

Theorem 19. *Let* $\Delta > \left\lfloor \frac{n \log n}{2 \log L} \right\rfloor$. *There exist* Δ*-restricted data graphs* $\mathcal{D}$ *over a vertex set of size* L, *such that any rewriting code* $\mathcal{C}$ *that allows the representation of the corresponding* Δ*-restricted data in* n *flash cells of* q *levels supports at most* $t(\mathcal{C}) = O\left(\frac{nq \log n}{\log \Delta}\right)$ *rewrites.*

Proof: We start by showing that Δ-restricted graphs $\mathcal{D}$ with certain properties do not allow rewriting codes $\mathcal{C}$ that support more than $t(\mathcal{C}) = O\left(\frac{nq \log n}{\log \Delta}\right)$ rewrites. We then show that such graphs do indeed exist. This will conclude our proof.

Let $\mathcal{D}$ be a Δ-restricted graph whose diameter d is at most $O\left(\frac{\log L}{\log \Delta}\right)$. Assuming the existence of such a graph $\mathcal{D}$, consider (by contradiction) a rewriting code $\mathcal{C}$, for the Δ-restricted data described by $\mathcal{D}$, that allows $t(\mathcal{C}) = \omega\left(\frac{nq \log n}{\log \Delta}\right)$ rewrites. We use $\mathcal{C}$ to construct a rewriting code $\mathcal{C}'$ for a new data graph $\mathcal{D}'$ which has the same vertex set $V_{\mathcal{D}'} = V_{\mathcal{D}}$ but is a complete graph. The code $\mathcal{C}'$ will allow $t(\mathcal{C}') = \omega\left(\frac{nq \log n}{\log L}\right)$ rewrites, a contradiction to Theorem 18. This will imply that our initial assumption regarding the quality of our rewriting code $\mathcal{C}$ is false.

The rewriting code $\mathcal{C}'$ (defined by the decoding function F'_d and the update function F'_u) is constructed by *mimicking* $\mathcal{C}$ (defined by the decoding function F_d and the update function F_u). We start by setting $F'_d = F_d$. Next, let s be some state of the flash cells. Denote $F_d(s) = F'_d(s) = v_0 \in V_{\mathcal{D}}$. Consider a rewrite operation attempting to store a new value $v_1 \in V_{\mathcal{D}}$, $v_1 \neq v_0$. There exists a path in $\mathcal{D}$ of length at most $d' \leqslant d$ from v_0 to v_1 which we denote by $v_0, u_1, u_2, \ldots, u_{d'-1}, v_1$. We now define

$$F'_u(s, v_1) = F_u(F_u(\ldots F_u(F_u(s, u_1), u_2) \ldots, u_{d'-1}), v_1),$$

which simply states that to encode a new value v_1 we follow the steps taken by the code $\mathcal{C}$ on a short path from v_0 to v_1 in the data graph $\mathcal{D}$.

As $\mathcal{C}$ allows $t(\mathcal{C}) = \omega\left(\frac{nq\log n}{\log\Delta}\right)$ rewrites, the code for $\mathcal{C}'$ allows at least

$$t(\mathcal{C}') = \omega\left(\frac{nq\log n}{d\log\Delta}\right) = \omega\left(\frac{nq\log n}{\log L}\right)$$

rewrites. Here we use the fact that $d = O\left(\frac{\log L}{\log\Delta}\right)$.

It is left to show the existence of data graphs $\mathcal{D}$ of maximum out-degree Δ whose diameter d is at most $O\left(\frac{\log L}{\log\Delta}\right)$. Such graphs exist for any $\Delta \geqslant \left\lfloor \frac{n\log n}{2\log L}\right\rfloor \geqslant \omega(\log^3 L)$ (recall our setting of $L \leqslant 2^{\sqrt{n}}$). Namely, consider the distribution $G_{L,p}$ over graphs $G = (V,E)$ in which $|V| = L$ and each pair (v_1,v_2) in $V\times V$ is chosen to be in E independently with probability p. For $pL = \omega(\log^3 L)$, in [3] (Chapter 10) it is shown that with high probability the maximum degree in G is $\Delta \leqslant 2pL$ and the diameter d of G is at most $2\frac{\log L}{\log pL}$. This implies the existence of graphs G with maximum degree Δ and diameter $O\left(\frac{\log L}{\log\Delta}\right)$ as desired. ∎

IV. Robust Code

When rewriting is a random process, it is interesting to design codes with good expected performance. We can also use randomized code constructions to improve the expected performance. In this section, we study two types of such codes, the *strongly robust codes* and the *weakly robust codes*. As before, we focus on the flash memory model, where n cells of q levels store the data from a data graph $\mathcal{D}$ of L vertices.

A *strongly robust code* is a randomized code that maximizes the expected number of supported rewrites for *every* rewriting sequence. In this section, we present a code such that for every rewriting sequence, the expected number of supported rewrites is $n(q-1) - o(nq)$. It is clearly strongly robust.

We define a *weakly robust code* to be a code that maximizes the expected number of supported rewrites for *every* rewriting model that follows an i.i.d. distribution, specified as follows. Let $\{0,1,\ldots,L-1\}$ denote the alphabet of the data. Let $p_0,p_1,\ldots,p_{L-1}$ be L *positive* probabilities such that $\sum_{i=0}^{L-1} p_i = 1$. Assume that events happen only at discrete times $t_1,t_2,t_3,\ldots$, and at time t_j (for $j = 1,2,3,\ldots$), the data follows an i.i.d. distribution: it has value i with probability p_i, for $i = 0,1,\ldots,L-1$. If at time t_j, the data changes to a value different from that of time t_{j-1}, then there is a rewrite. Clearly, if at some moment the data is i, the next rewrite will change it to $j \neq i$ with probability $p_j/\sum_{k\in\{0,\ldots,i-1,i+1,\ldots,L-1\}} p_k$. In this section, we present a *deterministic* code such that for any positive probability set $(p_0,p_1,\ldots,p_{L-1})$, the expected number of supported rewrites is $n(q-1)-o(nq)$. This code is clearly weakly robust.

A. Code Construction

In the trajectory code, the basic building block is a code whose data graph $\mathcal{D}$ is a complete graph and where $n \geqslant L$. In this section, we focus on robust codes with $n \geqslant L$. There is no restriction on their data graphs. The robust codes can be used as the building blocks in the trajectory code.

Let $(c_1,c_2,\ldots,c_n)$ denote the n cell levels in the flash memory model. Given $\vec{c} = (c_1,c_2,\ldots,c_n)$, define its *weight* $w(\vec{c})$ as $w(\vec{c}) = \sum_{i=1}^n c_i$. Clearly, $0 \leqslant w(\vec{c}) \leqslant (q-1)n$. Given the *decoding function* F_d, the cell state $\vec{c}$ represents some symbol $F_d(\vec{c}) \in \{0,1,\ldots,L-1\}$. We now present a code construction.

Construction 20. *Choose parameters $\theta_{i,j}$ and a_i from the set $\{0,1,\ldots,L-1\}$ for all $0 \leqslant i \leqslant n(q-1)-1$ and $1 \leqslant j \leqslant n$. The specific values of $\theta_{i,j}$ and a_i will determine the code's performance. Given a cell state $\vec{c} = (c_1,c_2,\ldots,c_n)$,*

$$F_d(\vec{c}) = \left(\sum_{i=1}^{n}\theta_{w(\vec{c})-1,i}c_i + \sum_{i=0}^{w(\vec{c})-1} a_i\right) \bmod L.$$

By default, if $\vec{c} = (0,0,\ldots,0)$, then $F_d(\vec{c}) = 0$.

For simplicity, we will omit the term "$\bmod L$" in all computations below that consist of values of data. For example, the formula in the above construction will be simply written as $F_d(\vec{c}) = \sum_{i=1}^n \theta_{w(\vec{c})-1,i}c_i + \sum_{i=0}^{w(\vec{c})-1} a_i$, and $F_d(\vec{c}) - F_d(\vec{c'})$ will mean $F_d(\vec{c}) - F_d(\vec{c'}) \bmod L$.

Given a cell state $\vec{c} = (c_1,\ldots,c_n)$, define its *$i$-th neighbor* as $N_i(\vec{c}) = (c_1,\ldots,c_{i-1},c_i+1,c_{i+1},\ldots,c_n)$ (provided that the cell state $N_i(\vec{c})$ exists), for $i = 1,\ldots,n$. There is a directed edge in the memory graph $\mathcal{M}$ from vertex $\vec{c}$ to vertex $N_i(\vec{c})$. Call this edge the *i-th outgoing edge of $\vec{c}$* and the *i-th incoming edge of $N_i(\vec{c})$*. Define $e_i(\vec{c}) = F_d(N_i(\vec{c})) - F_d(\vec{c})$, and call $e_i(\vec{c}) \in \{0,1,\ldots,L-1\}$ the *value* of this edge. Let $\psi(\vec{c}) = |\{e_i(\vec{c}) \mid i = 1,2,\ldots,n\}|$, and call $\psi(\vec{c})$ the *diversity* of $\vec{c}$. Note that $\psi(\vec{c})$ is the number of different values that the outgoing edges of $\vec{c}$ take. For efficient rewriting, it is beneficial for $\psi(\vec{c})$ to be large.

Lemma 21. *Let $\vec{c} = (c_1,\ldots,c_n)$ be a cell state such that $c_i < q-1$ for $i = 1,\ldots,n$. With Construction 20,*

$$\psi(\vec{c}) = \left|\left\{\theta_{w(\vec{c}),i} \mid i = 1,2,\ldots,n\right\}\right|.$$

Proof: We have

$$\begin{aligned} e_i(\vec{c}) &= F_d(N_i(\vec{c})) - F_d(\vec{c}) \\ &= \sum_{j=1}^{n}\theta_{w(\vec{c}),j}c_j + \theta_{w(\vec{c}),i} + \sum_{j=0}^{w(\vec{c})} a_j \\ &\quad - \sum_{j=1}^{n}\theta_{w(\vec{c})-1,j}c_j - \sum_{j=0}^{w(\vec{c})-1} a_j \\ &= \theta_{w(\vec{c}),i} + a_{w(\vec{c})} + \sum_{j=1}^{n}(\theta_{w(\vec{c}),j} - \theta_{w(\vec{c})-1,j})c_j \end{aligned}$$

Only the first term, $\theta_{w(\vec{c}),i}$, depends on i. Hence, $\psi(\vec{c}) = |\{e_i(\vec{c}) \mid i = 1,2,\ldots,n\}| = \left|\left\{\theta_{w(\vec{c}),i} \mid i = 1,2,\ldots,n\right\}\right|$. ∎

So to make $\psi(\vec{c})$ large, it is sufficient to choose parameters in Construction 20 such that $\left|\left\{\theta_{w(\vec{c}),i} \mid i = 1,2,\ldots,n\right\}\right|$ is large. We now analyze the robustness of the construction.

B. Strong Robustness

In this subsection, for succinctness, we analyze a simplified version of Construction 20. (The general construction can have more variations, and the analysis here can be readily used for it.) Assume $n \geqslant L$. For $i = 1, 2, \ldots, L$, define $g_i = \{j \mid 1 \leqslant j \leqslant n, j \equiv i \pmod{L}\}$. For example, if $n = 8, L = 3$, then $g_1 = \{1,4,7\}, g_2 = \{2,5,8\}, g_3 = \{3,6\}$. Also define $h_i = \sum_{j \in g_i} c_j$, where c_j is the j-th cell level. In the following construction, the cells in the same set g_i work as a "super cell."

Construction 22. *(*STRONGLY-ROBUST CODE*) For all $0 \leqslant i \leqslant n(q-1) - 1$, choose the parameter a_i* independently and uniformly at random *from $\{0, 1, \ldots, L-1\}$. Given a cell state $\vec{c} = (c_1, c_2, \ldots, c_n)$, set*

$$F_d(\vec{c}) = \sum_{i=1}^{L} i h_i + \sum_{i=0}^{w(\vec{c})-1} a_i.$$

For every rewrite, change the cells to a new state such that this new cell state represents the new data value and its weight is minimized. (If there is a tie, break it arbitrarily.)

The above code has a randomized construction that uses the random numbers $a_0, a_1, \ldots, a_{n(q-1)-1}$. These random numbers are stored in separate cells from the code, and are unrelated (that is, unknown) to the rewriting sequences. They are generated only once and can be used by many codes with the same construction, so their cost can be omitted.

Lemma 23. *Let $n \geqslant L$. Let $\vec{c} = (c_1, \ldots, c_n)$ be a cell state such that $h_i < (q-1)|g_i|$ for $i = 1, \ldots, L$. ($|g_i|$ is the cardinality of the set g_i.) With Construction 22, if $\vec{c}$ is the current cell state, no matter what value the next rewrite changes the data to, the rewrite increases the weight of the cell state only by one, and it increases h_i by one with probability $\frac{1}{L}$ for all $i \in \{1, \ldots, L\}$.*

Proof: From Construction 22, we can see that if we increase h_i by one, the data value will increase by $i + a_{w(\vec{c})}$ (modulo L), for $i \in \{1, \ldots, L\}$. Since $\{i + a_{w(\vec{c})} \mid i = 1, \ldots, L\} = \{0, 1, \ldots, L-1\}$, the rewrite will increase exactly one cell level by one. Since $a_{w(\vec{c})}$ is uniformly random over $\{0, 1, \ldots, L-1\}$, so is $i + a_{w(\vec{c})}$. So the rewrite will increase h_i by one with probability $\frac{1}{L}$. ∎

The above lemma applies to *all* rewriting sequences.

Theorem 24. *Let $L^2 \log L = o(qn)$, and $\frac{n \bmod L}{L} = o(1)$. For a code $\mathcal{C}$ of Construction 22, for every rewriting sequence, the expected number of rewrites it supports is $n(q-1) - o(nq)$.*

Proof: Consider L bins that can, respectively, contain $(q-1)|g_1|, (q-1)|g_2|, \ldots, (q-1)|g_L|$ balls. Use h_i to denote the number of balls in the i-th bin, and increasing h_i by one is the same as throwing a ball into the i-th bin. Note that every bin can contain at least $(q-1) \cdot \lfloor \frac{n}{L} \rfloor$ balls and at most $(q-1) \cdot \lceil \frac{n}{L} \rceil$ balls. By Lemma 23, before any bin is full, a rewrite throws a ball uniformly at random into the L bins. The rewriting process can always continue before any bin becomes full. Thus, the number of rewrites supported by the code $\mathcal{C}$ is at least the number of balls thrown to make at least one bin full.

Suppose that $n(q-1) - c\sqrt{n(q-1)}$ balls are uniformly randomly thrown into L bins, and there is no limit on the capacity of any bin. Here c is sufficiently large and $L^2 \log L = o(c^2)$, $c^2 = o(qn)$. For $i = 1, \ldots, L$, let x_i denote the number of balls thrown into the i-th bin. Clearly, $E[x_i] = \frac{n(q-1)}{L} - \frac{c\sqrt{n(q-1)}}{L}$. By the Chernoff bound, when nq is sufficiently large, the probability that the i-th bin contains more than $(q-1) \cdot \lfloor \frac{n}{L} \rfloor$ balls is less than $e^{-\Omega(c^2/L^2)}$. By the union bound, the probability that one or more of the L bins contain more than $(q-1) \cdot \lfloor \frac{n}{L} \rfloor$ balls is less than $Le^{-\Omega(c^2/L^2)}$. Since c is sufficiently large and $L^2 \log L = o(c^2)$, $Le^{-\Omega(c^2/L^2)} = o(1)$.

So when $n(q-1) - c\sqrt{n(q-1)}$ balls are uniformly randomly thrown into L bins, with high probability, all the L bins have $(q-1) \cdot \lfloor \frac{n}{L} \rfloor$ or fewer balls. Since $n(q-1) - c\sqrt{n(q-1)} = n(q-1) - o(nq)$, we get the conclusion. ∎

Note that the number of rewrites can never exceed $n(q-1)$. Theorem 24 shows that asymptotically (in q and n), Construction 22 is strongly robust under mild conditions.

C. Weak Robustness

We now consider a deterministic version of Construction 20.

Construction 25. *(*WEAKLY ROBUST CODE*) Given a cell state $\vec{c} = (c_1, c_2, \ldots, c_n)$,*

$$F_d(\vec{c}) = \sum_{i=1}^{L} i h_i + \sum_{i=0}^{w(\vec{c})-1} i.$$

For every rewrite, change the cells to a new state such that this new cell state represents the new data value and its weight is minimized. (If there is a tie, break it arbitrarily.)

Theorem 26. *Let $L \geqslant 3$ be a constant and let n be a multiple of L. For a code $\mathcal{C}$ of Construction 25, for any i.i.d. rewriting model with a positive probability set $(p_0, p_1, \ldots, p_{L-1})$, the expected number of rewrites it supports is $n(q-1) - o(nq)$.*

Proof: For $i = 1, 2, \ldots, L$, let us see the cells in the set g_i as a "super cell" of $(q-1)|g_i| + 1$ levels. Then we can see $\vec{c} = (h_1, h_2, \ldots, h_L)$ as the state of these L super cells, where h_i is the level of the i-th super cell. For $i = 1, 2, \ldots, L$, if we increase h_i by one, the data's value will be increased by $(\sum_{j=1}^{L} j h_j + i + \sum_{j=0}^{w(\vec{c})} j) - (\sum_{j=1}^{L} j h_j + \sum_{j=0}^{w(\vec{c})-1} j) = i + w(\vec{c}) \pmod{L}$. Let us call $u(\vec{c}) = (1 + w(\vec{c}), 2 + w(\vec{c}), \ldots, L + w(\vec{c}))$ the *update vector* of the super-cell state $\vec{c} = (h_1, h_2, \ldots, h_L)$. Before any super-cell reaches its highest level, a rewrite will increase the weight of the cell state by only one. So it is easy to see that initially, $\vec{c} = (0, 0, \ldots, 0)$ and $u(\vec{c}) = (1, 2, \ldots, L-1, 0)$; after one rewrite, $u(\vec{c}) = (2, 3, \ldots, L-1, 0, 1)$; after the second rewrite, $u(\vec{c}) = (3, 4, \ldots, L-1, 0, 1, 2)$; and so on. After exactly L rewrites, $u(\vec{c}) = (1, 2, \ldots, L-1, 0)$ again. The update vector shifts cyclically with a period of L.

We model the rewriting process as a Markov chain as follows. Every state in the Markov chain is represented by

a pair $(\mathcal{V};\vec{u})$, where $\mathcal{V}\in\{0,1,\ldots,L-1\}$ is the value of the data at a given time, and $\vec{u}$ is the update vector of the super-cell state at that same time. Since $\mathcal{V}$ can take L values and $\vec{u}$ can take L values, there are totally L^2 states in the Markov chain. A state can transit to another state iff a rewrite can make such a change happen. (We assume that no super-cell has any upper bound on its cell level.) A state $(\mathcal{V};\vec{u})$ can transit to the state $(\mathcal{V}';\vec{u}')$ if and only if $\mathcal{V}'\neq\mathcal{V}$ and $\vec{u}'$ is the next cyclic shift of $\vec{u}$, and the transit probability is $\frac{p_{\mathcal{V}'}}{\sum_{i\in\{0,1,\ldots,L-1\}-\{\mathcal{V}\}}p_i}>0$. Clearly, such a Markov chain is irreducible, positive recurrent, and has a period of L (because the update vectors have a period of L).

Let $z\to\infty$ be a positive integer. We consider a sequence of $\lceil\sqrt{z}\rceil L+zL^2+L$ rewrites, which we divide into $L+1$ pieces. The first piece is the first $\lceil\sqrt{z}\rceil L$ rewrites. For $i=2,3,\ldots,L+1$, the i-th piece is the following $zL+1$ rewrites. We consider the rewrite sequences from the given i.i.d. rewriting model. Clearly, as z increases, the length of these $\lceil\sqrt{z}\rceil L+zL^2+L$ rewrites also increases. It is easy to see that for $i=2,3,\ldots,L+1$, before the first rewrite of the i-th piece happens, the update vector is $(1+i-2,2+i-2,\ldots,L+i-2)$. So the L update vectors at the beginning of these L pieces of rewrites are all different.

The value of the data has a stationary distribution in the Markov chain, which is $(p_0,p_1,\ldots,p_{L-1})$. So no matter what the initial value of the data is, after many rewrites, it converges to this stationary distribution. Now consider the second piece of rewrites. Before the first rewrite of the second piece starts, as $z\to\infty$, the Markov chain is in the state $(i;(1,2,\ldots,L))$ with probability p_i. Every transition in the Markov chain corresponds to a rewrite, and the super-cell whose level is increased by this transition is uniquely determined by the two Markov-chain states determining this transition. So corresponding to the $zL+1$ rewrites in the second piece of rewrites, the percentage that each transition happens also converges. This implies the existence of a vector $(x_1,x_2,\ldots,x_L)$ such that $\sum_{i=1}^{L}x_i=zL+1$ and for $i=1,\ldots,L$, the i-th super-cell's level, h_i, is increased $x_i+o(x_i)$ times during the second piece of rewrites with high probability.

For $j=2,3,\ldots,L+1$, before the first rewrite of the j-th piece of rewrites starts, the Markov chain is in the state $(i;(1+j-2,2+j-2,\ldots,L+j-2))$ with probability p_i. By symmetry, for $i=1,\ldots,L$, h_i is increased $x_{i+(j-2)\bmod L}+o(x_{i+(j-2)\bmod L})$ times with high probability. (In the previous expression, if $i+(j-2)=L$, then let $x_{i+(j-2)\bmod L}$ be x_L.) So over the zL^2+L rewrites of the 2nd, 3rd, $\ldots$, $L+1$-th pieces of rewrites, h_i is increased $\sum_{j=1}^{L}x_j+o(\sum_{j=1}^{L}x_j)=zL+o(zL)$ times with high probability. Since $\lceil\sqrt{z}\rceil=o(z)$, over the whole rewriting sequence (of $L+1$ pieces), h_i is increased $zL+o(zL)$ times with high probability. Since every super-cell has a maximum level of $\frac{n(q-1)}{L}$, the number of rewrites it takes for any super-cell to reach its maximum level is $n(q-1)-o(nq)$ with high probability. So the expected number of rewrites that the code $\mathcal{C}$ supports is $n(q-1)-o(nq)$. ■

The above theorem shows the weak robustness of the code construction.

V. Concluding Remarks

In this paper, we present the trajectory code for rewriting and show its optimality. We also present a robust code construction for the optimization of the expected rewriting performance. It will be interesting to study more constrained memory models and rewriting models, and also combine error correction with the rewriting codes. That remains as our future research.

Acknowledgment

This work was supported in part by the NSF CAREER Award CCF-0747415, the NSF grant ECCS-0802107, the ISF grant 480/08, the GIF grant 2179-1785.10/2007, and the Caltech Lee Center for Advanced Networking.

References

[1] R. Ahlswede and Z. Zhang, "On multiuser write-efficient memories," *IEEE Trans. on Inform. Theory*, vol. 40, no. 3, pp. 674–686, 1994.

[2] V. Bohossian, A. Jiang, and J. Bruck, "Buffer coding for asymmetric multi-level memory," in *Proceedings of the 2007 IEEE International Symposium on Information Theory (ISIT2007), Nice, France*, Jun. 2007, pp. 1186–1190.

[3] B. Bollobás, *Random Graphs (2nd Edition)*. Cambridge University Press, 2001.

[4] P. Cappelletti, C. Golla, P. Olivo, and E. Zanoni, *Flash memories*. Kluwer Academic Publishers, 1999.

[5] G. D. Cohen, P. Godlewski, and F. Merkx, "Linear binary code for write-once memories," *IEEE Trans. on Inform. Theory*, vol. IT-32, no. 5, pp. 697–700, Sep. 1986.

[6] A. Fiat and A. Shamir, "Generalized "write-once" memories," *IEEE Trans. on Inform. Theory*, vol. IT-30, no. 3, pp. 470–480, May 1984.

[7] H. Finucane, Z. Liu, and M. Mitzenmacher, "Designing floating codes for expected performance," in *Proc. of the Annual Allerton Conference*, 2008.

[8] F.-W. Fu and A. J. Han Vinck, "On the capacity of generalized write-once memory with state transitions described by an arbitrary directed acyclic graph," *IEEE Trans. on Inform. Theory*, vol. 45, no. 1, pp. 308–313, Jan. 1999.

[9] F. Fu and R. W. Yeung, "On the capacity and error-correcting codes of write-efficient memories," *IEEE Trans. on Inform. Theory*, vol. 46, no. 7, pp. 2299–2314, Nov. 2000.

[10] A. J. Han Vinck and A. V. Kuznetsov, "On the general defective channel with informed encoder and capacities of some constrained memories," *IEEE Trans. on Inform. Theory*, vol. 40, no. 6, pp. 1866–1871, 1994.

[11] C. D. Heegard, "On the capacity of permanent memory," *IEEE Trans. on Inform. Theory*, vol. IT-31, no. 1, pp. 34–42, Jan. 1985.

[12] C. D. Heegard and A. A. E. Gamal, "On the capacity of computer memory with defects," *IEEE Trans. on Inform. Theory*, vol. IT-29, no. 5, pp. 731–739, Sep. 1983.

[13] A. Jiang, V. Bohossian, and J. Bruck, "Floating codes for joint information storage in write asymmetric memories," in *Proceedings of the 2007 IEEE International Symposium on Information Theory (ISIT2007), Nice, France*, Jun. 2007, pp. 1166–1170.

[14] A. Jiang and J. Bruck, "Joint coding for flash memory storage," in *Proceedings of the 2008 IEEE International Symposium on Information Theory (ISIT2008), Toronto, Canada*, Jul. 2008, pp. 1741–1745.

[15] A. V. Kuznetsov and B. S. Tsybakov, "Coding for memories with defective cells," *Problemy Peredachi Informatsii*, vol. 10, no. 2, pp. 52–60, 1974.

[16] H. Mahdavifar, P. H. Siegel, A. Vardy, J. K. Wolf and E. Yaakobi, "A nearly optimal construction of flash codes," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, Seoul, Korea, June-July 2009.

[17] F. Merkx, "WOM codes constructed with projective geometries," *Traitment du Signal*, vol. 1, no. 2-2, pp. 227–231, 1984.

[18] R. L. Rivest and A. Shamir, "How to reuse a "write-once" memory," *Inform. and Control*, vol. 55, pp. 1–19, 1982.

[19] G. Simonyi, "On write-unidirectional memory codes," *IEEE Trans. on Inform. Theory*, vol. 35, no. 3, pp. 663–667, May 1989.
[20] W. M. C. J. van Overveld, "The four cases of write unidirectional memory codes over arbitrary alphabets," *IEEE Trans. on Inform. Theory*, vol. 37, no. 3, pp. 872–878, 1991.
[21] F. M. J. Willems and A. J. Vinck, "Repeated recording for an optical disk," in *Proc. 7th Symp. Inform. Theory in the Benelux*, May 1986, Delft Univ. Press, pp. 49-53.
[22] J. K. Wolf, A. D. Wyner, J. Ziv, and J. Korner, "Coding for a write-once memory," *AT&T Bell Labs. Tech. J.*, vol. 63, no. 6, pp. 1089–1112, 1984.
[23] E. Yaakobi, P. H. Siegel, and J. K. Wolf, "Buffer codes for multi-level flash memory," in *Proceedings of the 2008 IEEE International Symposium on Information Theory (ISIT2008), Toronto, Canada*, 2008, poster.
[24] E. Yaakobi, A. Vardy, P. H. Siegel, and J. K. Wolf, "Multidimensional flash codes," in *Proc. of the Annual Allerton Conference*, 2008.
[25] G. Zémor and G. Cohen, "Error-correcting WOM-codes," *IEEE Trans. on Inform. Theory*, vol. 37, no. 3, pp. 730–734, May 1991.