

# 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:  $0, 1, \dots, q-1$ . 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 [9]. 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  $(c_1, c_2, \dots, c_n)$ , where  $c_i \in \{0, 1, \dots, q-1\}$  is the  $i$ -th cell level, for  $i = 1, \dots, n$ . There is a directed edge from

$(c_1, c_2, \dots, c_n)$  to  $(c'_1, c'_2, \dots, c'_n)$  iff there exists exactly one index  $i \in \{1, \dots, n\}$  such that  $c'_i = c_i + 1$  while  $c'_j = c_j$  for  $j = 1, \dots, i-1, i+1, \dots, n$ .

**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 [9]) The data consists of  $k$  variables, each of which takes its value from the alphabet  $\{0, 1, \dots, \ell-1\}$ . Every rewrite changes the value of one variable. Hence, the data graph  $\mathcal{D}$  has  $L = \ell^k$  vertices, each of incoming and outgoing degree  $k(\ell-1)$ . The floating code model reduces to the write-once memory (WOM) code model [15] 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  $\ell$ .

**Definition 5.** (CODE FOR REWRITING) A code for rewriting has a decoding function  $F_d$  and an update function  $F_u$ . The decoding function  $F_d : V_{\mathcal{M}} \rightarrow V_{\mathcal{D}}$  maps a memory state  $s \in V_{\mathcal{M}}$  to the stored data  $F_d(s) \in V_{\mathcal{D}}$ . The update function (which represents a rewriting operation),  $F_u : V_{\mathcal{M}} \times V_{\mathcal{D}} \rightarrow V_{\mathcal{M}}$ , 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  $\mathcal{M}$ .

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.<sup>1</sup> We present a novel code construction, the *trajectory code*, based on tracing the changes of data in the

<sup>1</sup>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.

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  $r \leq 1$ , 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.

Due to the space limitation, we skip some details in multiple places. Interested readers are referred to [11].

## II. OVERVIEW OF RELATED RESULTS

There has been distinguished theoretical study on constrained memories. They include defective memories [12], write once memory (WOM) [15], write unidirectional memory (WUM) [16], [17], and write efficient memory [1]. Among them, WOM is the most related to the flash memory model studied in this paper. 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 codes have been presented over the years [5] [6] [14] [15]. (For a more detailed survey, please see [11].) 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], [9]. The rewriting schemes include floating codes [9], [10] 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 [9], [10], the flash codes in [13], [19], and the constructions based on Gray codes in [7] that optimize 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.

TABLE I

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

TYPE	ASYMPTOTIC OPTIMALITY	REF.
WOM code ( $\mathcal{D}$ is a complete graph)	$t(\mathcal{C})$ is asymptotically optimal	[15]
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 $\ell = \Theta(1)$	[9] [10]
floating code ( $\mathcal{D}$ is a hypercube)	$t(\mathcal{C})$ is asymptotically optimal when $n = \Omega(k \log k)$ and $\ell = \Theta(1)$	[9] [10]
floating code ( $\mathcal{D}$ is a hypercube)	$t(\mathcal{C})$ is asymptotically optimal when $n = \Omega(k^2)$ and $\ell = \Theta(1)$	[19]
buffer code ( $\mathcal{D}$ is a de Bruijn graph)	$t(\mathcal{C})$ is asymptotically optimal when $n = \Omega(k)$ and $\ell = \Theta(1)$	[2] [18]
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 $\Delta$ . 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

### 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,  $1 \leq i \leq d$ , 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 edges have unique labels from  $\{1, \dots, \Delta\}$ , where  $\Delta$  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  $S_1, \dots, S_d$ .

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  $s \bmod (d + 1)$ . 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  $(s-1) \bmod (d+1)$  edges which are read consecutively from  $S_1, S_2, \dots$ . 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  $i = 0, \dots, d$ , can be seen as a *smaller rewriting code* whose data graph is a *complete graph* of either  $L$  vertices (for  $S_0$ ) or  $\Delta$  vertices (for  $S_1, \dots, S_d$ ). 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  $L \geq 2$ , 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  $\Delta$ .)

We first state a scheme that allows approximately  $nq/8$  rewrites in the case in which  $2 \leq L \leq n$ . 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  $2 \leq L \leq n$* : In this section we present a code for small values of  $L$ . The code we present is essentially the one presented in [15].

**Construction 6.** Let  $2 \leq L \leq n$ . 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  $\vec{c} = (c_0, c_1, \dots, c_{L-1})$ , where  $c_i \in \{0, 1, \dots, q-1\}$  is the level of the  $i$ -th cell for  $i = 0, 1, \dots, L-1$ . Denote the alphabet of data by  $V_{\mathcal{D}} = \{0, 1, \dots, L-1\}$ . We first use only cell levels 0 and 1, and the data stored in the cells is  $\sum_{i=0}^{L-1} ic_i \pmod{L}$ . 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  $n \geq L$  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 7.** Let  $2 \leq L \leq n$ . The code  $\mathcal{C}$  in Construction 6 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  $j = 1, \dots, q-1$ ), by the analysis in [15], 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  $n \geq L$ . When  $b = \lfloor n/L \rfloor$ , it is easy to see that  $bL \geq n/2$ . The  $b$  groups of cells can guarantee  $t(\mathcal{C}) = \frac{b(L+4)(q-1)}{4} \geq \frac{n(q-1)}{8} = \Theta(nq)$  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  $n < L \leq 2\sqrt{n}$ . We will address the general case at the end of this section.

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

**Claim 8.** For  $16 \leq n \leq L \leq 2\sqrt{n}$  it holds that  $b \leq \frac{2 \log L}{\log n}$ .

**Construction 9.** Let  $n < L \leq 2\sqrt{n}$ . 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  $i = 1, 2, \dots, b$ , let  $v_i$  be a symbol from an alphabet of size  $\lfloor n/b \rfloor \geq L^{1/b}$ . We may represent any symbol  $v \in V_{\mathcal{D}}$  as a vector of symbols  $(v_1, v_2, \dots, v_b)$ . 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 6 to store  $v_i$ .

**Theorem 10.** Let  $16 \leq n \leq L \leq 2\sqrt{n}$ . The code  $\mathcal{C}$  in Construction 9 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 9, the number of rewrites possible is bounded by the number of rewrites possible for each of the  $b$  cell groups. By Theorem 7 and Claim 8, this is at least  $\lfloor \frac{n}{b} \rfloor \cdot \frac{q-1}{8} \geq \left(\frac{n \log n}{2 \log L} - 1\right) \frac{q-1}{8} = \Theta\left(\frac{nq \log n}{\log L}\right)$ . ■

### 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  $\Delta$ . We refer to such graphs as  $\Delta$ -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  $n \leq L \leq 2\sqrt{n}$ . Notice that for  $L \leq n$ , Construction 6 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  $\Delta$  is *small* compared to  $n$ , and the case in which  $\Delta$  is *large*.

**Construction 11.** Let  $\Delta \leq \left\lfloor \frac{n \log n}{2 \log L} \right\rfloor$ . We build an efficiently encodable and decodable rewriting code  $\mathcal{C}$  for any  $\Delta$ -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  $n_i = \lfloor n/(2d) \rfloor \geq \Delta$  for  $i = 1, \dots, d$ . (We obviously have  $\sum n_i \leq n$ .)

The update and decoding functions of the trajectory code  $\mathcal{C}$  are defined as follows: Consider using the encoding scheme specified in Construction 9 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 6 for the encoding of one of  $\{1, \dots, \Delta\}$  in the flash cells of  $S_i$  ( $i = 1, \dots, d$ ). Notice that the latter is possible as  $n_i \geq \Delta$  for  $i = 1, \dots, d$ .

**Theorem 12.** Let  $\Delta \leq \left\lfloor \frac{n \log n}{2 \log L} \right\rfloor$ . The code  $\mathcal{C}$  of Construction 11 guarantees  $t(\mathcal{C}) = \Theta(nq)$  rewrites.

*Proof:* By Theorems 10 and 7, the number of rewrites possible in  $S_0$  is equal (up to constant factors) to that of  $S_i$  ( $i \geq 1$ ):

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

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 13.** Let  $\left\lfloor \frac{n \log n}{2 \log L} \right\rfloor \leq \Delta \leq L$ . We build an efficiently encodable and decodable rewriting code  $\mathcal{C}$  for any  $\Delta$ -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  $i = 1, \dots, d$ .

The update and decoding functions of the trajectory code  $\mathcal{C}$  are defined as follows: Consider using the encoding scheme specified in Construction 9 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  $\{1, \dots, \Delta\}$  in the flash cells of  $S_i$  ( $i = 1, \dots, d$ ).

**Theorem 14.** Let  $\left\lfloor \frac{n \log n}{2 \log L} \right\rfloor \leq \Delta \leq L$ . The code  $\mathcal{C}$  of Construction 13 guarantees  $t(\mathcal{C}) = \Theta\left(\frac{nq \log n}{\log \Delta}\right)$  rewrites.

*Proof:* By Theorem 10, 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  $S_i$  ( $i \geq 1$ ):  $\Theta\left(\frac{n_i q \log n_i}{\log \Delta}\right) = \Theta\left(\frac{nq \log n}{d \log \Delta}\right) = \Theta\left(\frac{nq \log n}{\log L}\right)$ . Here we use the fact that as  $d \leq \log L$  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 12, 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. We first note that any rewriting code  $\mathcal{C}$  that stores symbols from some data graph  $\mathcal{D}$  in  $n$  flash cells of  $q$  levels supports at most  $t(\mathcal{C}) \leq n(q - 1) = O(nq)$  rewrites (as each rewrite increases at least 1 cell state by at least 1). For large values of  $L$ , we can improve the upper bound. First, let  $r$  denote the largest integer such that  $\binom{r+n-1}{r} < L - 1$ .

**Claim 15.** For all  $1 \leq n < L - 1$ , it holds that  $r \geq \left\lfloor \frac{\log(L-1)}{1+\log n} \right\rfloor$ .

**Theorem 16.** 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

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

■

**Theorem 17.** Let  $\Delta > \left\lfloor \frac{n \log n}{2 \log L} \right\rfloor$ . There exist  $\Delta$ -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  $\Delta$ -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.

The proof of Theorem 17 appears in [11]. To prove our theorem we use  $\Delta$ -restricted graphs  $\mathcal{D}$  whose diameter  $d$  is at most  $O\left(\frac{\log L}{\log \Delta}\right)$  (see, e.g., Chapter 10 of [3]).

#### IV. ROBUST CODE

In this section, we study codes that optimize the expected rewriting performance. As before, we focus on the flash memory model, where  $n$  cells of  $q$  levels are used to store the data from a data graph  $\mathcal{D}$  of  $L$  vertices. We define a *strongly robust code* to be 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

