Designing Floating Codes for Expected Performance

Flavio Chierichetti, Hilary Finucane, Zhenming Liu, and Michael Mitzenmacher, *Member, IEEE*

***Abstract*—Floating codes are codes designed to store multiple values in a Write Asymmetric Memory, with applications to flash memory. In this model, a memory consists of a block of n cells, with each cell in one of q states $\{0, 1, \ldots, q-1\}$. The cells are used to represent k variable values from an ℓ-ary alphabet. Cells can move from lower values to higher values easily, but moving any cell from a higher value to a lower value requires first resetting the entire block to an all 0 state. Reset operations are to be avoided; generally a block can only experience a large but finite number of resets before wearing out entirely. A code here corresponds to a mapping from cell states to variable values, and a transition function that gives how to rewrite cell states when a variable is changed. Previous work has focused on developing codes that maximize the worst-case number of variable changes, or equivalently cell rewrites, that can be experienced before resetting. In this paper, we introduce the problem of maximizing the *expected* number of variable changes before resetting, given an underlying Markov chain that models variable changes. We demonstrate that codes designed for expected performance can differ substantially from optimal worst-case codes, and suggest constructions for some simple cases. We then study the related question of the performance of random codes, again focusing on the issue of expected behavior.**

***Index Terms*—Floating codes, average-case analysis, write asymmetric memory, random floating codes.**

I. Introduction

A LONG-STANDING albeit not widely studied subfield of coding theory involves data storage in a setting where the stored information can change state in only limited ways. The seminal and perhaps canonical example is the write-once memory introduced by Rivest and Shamir [21]. Motivated primarily by the potential of digital optical disks, the authors consider the setting of write-once bit positions, which they dub *wits*. (The name, apparently, has not lasted the test of time.) Each wit initially contains a 0 that may subsequently be rewritten as a 1, but such a write is irreversible; a 1 cannot later be changed back to a 0. Paper punch cards provide a useful example of wits; once a position is punched, it cannot be reset.

If permanent storage is required, a medium of wits is quite useful, but if rewriting of information is required, this irreversibility is problematic. Rivest and Shamir therefore consider the question of how many wits are required to allow t rewrites of a k-bit value. By carefully choosing how wits can be used to represent values, they are able to design schemes that do significantly better than the naïve scheme using kt wits.

A variety of related models have been considered over the years; see, for example, [1], [3]–[5], [7], [8], [10]–[14], [16], [22], [23]. Questions regarding coding schemes of this type have resurfaced in recent years with the introduction of flash memories, which work under similar principles [13]. A flash memory utilizes floating-gate cells, which can be modeled as having q states $\{0, 1, 2, \ldots, q-1\}$. Roughly speaking, the states correspond to the number of electrons being held by the cell. Adding electrons to a cell is easy, but removing electrons from a cell is difficult. In our model, that means it is easy to move a cell from a lower-numbered state to a higher-numbered one, but not the other way around. Indeed, cells are generally organized into blocks, and in order to lower a state value within a cell, one must reset an entire block back to the all 0 state. Resetting blocks is considered very expensive, first because the rewriting time when resetting a block is large, but perhaps more importantly because the lifetime of a flash memory generally only allows a large but essentially fixed number of reset operations before the memory is no longer usable [2].

Previous work, including recent work on codes for flash memories, has focused on the problem of maximizing the number of rewrite operations before a reset operation is required *in the worst case*. Codes for this setting were dubbed floating codes [13] (although it was suggested that they be called flash codes in [24]; we utilize the former). Worst-case analysis is certainly a natural approach, particularly in settings where no resets are possible, and there is a need for fixed guarantees on rewrite performance in unknown environments. We suggest, however, that in the setting of flash memories, where the product will be mass-produced, the product lifetime may allow a large number of reset operations, and there is the potential to study and model user behavior, *statistical* performance guarantees are more appropriate. The idea of average-case performance of codes in this setting is not new; it is explicitly mentioned by Rivest and Shamir as an open question [21]. (Also, see very different notions of average-case analysis in [1] and [16].) As far as we know, however, our work actually initiates the study of the *expected* performance of floating codes. This line of work has since been followed up

Manuscript received February 14, 2009; revised August 06, 2009. Current version published March 10, 2010. The work of Z. Liu was supported in part by the NSF by Grant CCF-0634923. The work of M. Mitzenmacher was supported in part by the NSF by Grant CCF-0634923 and also his work was done while visiting the School of Engineering and Applied Sciences, Harvard University. The material in this paper was presented in part at the 46th Annual Allerton Conference, Monticello, IL, October 2008.

F. Chierichetti is with the Dipartimento di Informatica, Sapienza University of Rome, Italy. He was with the School of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138 (e-mail: chierichetti@di.uniroma1.it).

H. Finucane was with the School of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138 USA (e-mail: hilaryfinucane@gmail.com).

Z. Liu and M. Mitzenmacher are with the School of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138 USA (e-mail: zliu@fas.harvard.edu; michaelm@eecs.harvard.edu).

Communicated by T. Etzion, Associate Editor for Coding Theory.

Digital Object Identifier 10.1109/TIT.2009.2039040

on by for example [15], which considers expected performance in a similar fashion.

Our primary contribution is a general model of the underlying problem of average-case performance of floating codes. Then, following the approach of [13], we consider particular possible implementations for specific parameter settings. This work should be seen as a starting point, providing building blocks for future work and raising a variety of challenging open problems to consider. Finally, returning to basic principles, we consider random codes in this setting, and consider both their worst-case and expected performance.

II. A General Model for Average-Case Performance of Floating Codes

We begin by reviewing the model for and definitions of floating codes, utilizing the framework of [13]. The memory stores k variable values from an ℓ-ary alphabet, given by the variable vector $(v_1, v_2, \ldots, v_k)$ with $v_i \in \{0, 1, \ldots, \ell - 1\}$. The memory consists of a block of n cells, represented by a cell state vector $(c_1, c_2, \ldots, c_n)$ with $c_i \in \{0, 1, \ldots, q - 1\}$. A cell state vector $(c_1, c_2, \ldots, c_n)$ is said to be *above* a cell state vector $(d_1, d_2, \ldots, d_n)$ if $c_i \geq d_i$ for all i. Abusing notation, we will write $x \geq y$ if x and y are cell state vectors such that x is above y, and $x \not\geq y$ if x is not above y. (We avoid vector notation where the meaning is clear.) For a cell state vector x to change to another state y with $y \not\geq x$, the memory must first be reset. If $y \geq x$, no reset is needed.

A floating code is defined by two functions. The decoding function $D : \{0, 1, \ldots, q - 1\}^n \to \{0, 1, \ldots, \ell - 1\}^k$ maps cell state vectors to variable vectors, and is used to decode the memory, transforming its current contents to the current variable values. The rewriting function $R : \{0, 1, \ldots, q - 1\}^n \times \{1, \ldots, k\} \times \{0, 1, \ldots, \ell - 1\} \to \{0, 1, \ldots, q - 1\}^n$ gives information on how to transition when a single variable value changes; given a current cell state vector, a variable value to be changed, and the value that variable is changed to, the function R gives a new corresponding cell state vector. The restriction on R is that the current cell state vector must always decode via D to the current variable vector. Here when a rewrite causes a transition from x to y with $y \not\geq x$, the cost is 1 due to a reset; otherwise, the cost is 0.

(We remark that in other work the rewriting function has included a special range value to denote a reset, or erasure, operation, which essentially terminates the process. Because we consider a stochastic framework that can involve many resets, we find it clearer to not include a specific reset token, but instead use the notion of transition costs to denote conditions that would cause a reset. This also provides a framework that more easily generalizes to allow variable costs.)

The goal of [13] and [24] is to find decoding and rewriting functions that maximize (starting from the all-zero vector) the number of rewrites before a reset *in the worst case*. We here consider a different goal. We assume that there is a Markov chain with state space $\{0, 1, \ldots, \ell - 1\}^k$ describing the behavior of the variable vector. For the most part we follow [13] and assume that only one variable changes at each transition, although more general Markov chains and rewriting functions can be considered, as we discuss below. Given a decoding function D and rewriting function R, the Markov chain on the variable vector induces a corresponding Markov chain on the cell state vector. Let the equilibrium distribution of this chain be given by π_x, and let p_{xy} be the probability of transitioning to state vector y when in a state vector x. Then using standard theory, the average long-term cost per variable change is given by

$$A = \sum_{y \not\geq x} \pi_x p_{xy}. \tag{1}$$

Also, by standard renewal theory $1/A$ can be considered the long-term average time between reset operations. Our goal is to find functions D and R that minimize this cost A.

We emphasize that the above model can obviously be generalized in many directions, in manners similar to other proposed generalizations of simple write-only memories. We may have arbitrary alphabets $\mathcal{V}$ for the variables and $\mathcal{S}$ for the states, with an arbitrary stochastic process on $\mathcal{V}^k$ inducing corresponding processes on $\mathcal{S}^n$, given the rewriting transition function. The rewriting function could allow multiple variables to change at once, taking the form $R : \{0, 1, \ldots, q - 1\}^n \times \{0, 1, \ldots, \ell - 1\}^k \to \{0, 1, \ldots, q - 1\}^n$, mapping the current cell state and a new value vector to a new cell state. There may be rules that a priori limit the transitions possible under the function R. There may be costs associated with all possible transitions (and/or all possible state vectors), and more general functions of these costs could be optimized. There may be history associated with either the variable state or the cell state, which could be incorporated into the decoding and rewriting functions. Many other generalizations can be imagined.

Indeed, we see the full generality of this coding problem as having potential applications beyond coding theory. Notice that, given the decoding function D, we can view the rewriting function R as a policy for a Markov decision process on the cell state vector, where the possible actions at a state correspond to the collection of transitions to be made for each possible state change. Hence, underlying this problem is the question of *how to design* a Markov decision process in the setting where we have an environment (the variable process) and we wish to minimize some function on the induced Markov decision process (the average cost per transition of the cell state process), where we have the ability to design the Markov decision process according to certain rules (in this case, the rules are that decoding is always successful). We are not aware of this specific problem in previous literature, and although we have no complexity results, we conjecture that many variations of this more general problem are at least NP-hard. (For a related NP-hardness result, see [5].)

III. The Case of $k = 2, \ell = 2$

A. The Case of $k = 2, \ell = 2, n = 2$

We begin with the case where $k = 2, \ell = 2$, and $n = 2$. That is, our value vector consists of two bits, and our states consist of two values in $\{0, \ldots, q - 1\}$. Optimal codes under worst case analysis for these codes were described in [13]; they guarantee $(q - 1) + \lfloor \frac{q-1}{2} \rfloor \approx \frac{3}{2}(q - 1)$ transitions before a reset. Here we consider asymptotically optimal codes (as q grows large) for the average case, under the simple but natural model where the

00	01	11	10
10	00	01	11
11	10	00	01
01	11	10	00

Fig. 1. The code, or the 2DGC, for $k = 2, \ell = 2, n = 2, q = 4$. The upper left corner represents the decoded value vector for cell (0,0); the lower right corner represents the decoded value vector for cell (3,3). Transitions are greedy, to the closest available cell decoding to the appropriate value.

00	01	11	10	00	01	11	10
10	00	01	11	10	00	01	11
11	10	00	01	11	10	00	01
01	11	10	00	01	11	10	00
00	01	11	10	00	01	11	10
10	00	01	11	10	00	01	11
11	10	00	01	11	10	00	01
01	11	10	00	01	11	10	00

Fig. 2. Code for $k = 2, \ell = 2, n = 2, q = 8$.

first of the two value bits is the next to flip with fixed probability p and the second is the next to flip with probability $1-p$ throughout the process. By asymptotically optimal, we mean that when $k = \ell = 2$ and n is fixed, the expected number of moves before a reset grows like $n(q-1) - o(q)$. (Indeed, we show this is the number of moves with high probability, with respect to the parameter q.) While it is perhaps not surprising that one could find an asymptotically optimal code for a given p, we show that in fact there exist codes that are simultaneously asymptotically optimal for every $p, 0 \le p \le 1$.

For convenience, we point out that in this setting we can adapt the notation of [24], who for the worst-case setting define the *deficiency* to be $n(q-1)-t$, where t is the number of guaranteed writes before a reset (starting from the all 0 state). We will use the term deficiency in a similar manner, although in our setting the deficiency will be random variable; a code with n cells will on some input achieve a deficiency of t if $n(q-1)-t$ transitions are performed before a reset is necessary on that input.

Our code systems are based on *Gray codes* [9]. While we give codes for all values of n, we begin with the important case of $n = 2$. We represent our code pictorially, for the case of $q = 4$, as follows (Fig. 1).

The upper left hand corner represents the decoded value for the cell state (0,0); it is written as 00, representing that both value bits are 0. More generally, the value in the ith row and jth column (counting from 0) represents the value vector under D for the cell state vector (i, j). Hence, for example, the decoded value for cell state (2,1) is 10 (first bit 1, second bit 0). The rewriting function is implicit; from each cell, one moves to the closest available cell (generally one space *down* or to the *right*) that decodes to the proper new value, and a reset occurs only if no move is possible in the down and rightward direction. Note that on a reset, one starts the next cycle in one of three positions: (0,1),(1,0) or (2,0). (Because of the 00 in the lower right hand corner, one can never start after a reset at (0,0)).

For larger values of q, we simply cycle through the Gray code values repeatedly. For example, the code when $q = 8$ is given in (Fig. 2).

We call these 2-dimensional Gray Codes, or a 2DGC for short. As we shall see, the 2DGC is not optimal, but it is asymptotically optimal, simultaneously for every value of p, in the following sense.

Theorem 1: For any $p \ge 1/2$ and $\delta > 0$, for any $q \ge q_0$ (where q_0 may depend on p and δ) the deficiency for the 2DGC is at most $4\lceil\sqrt{\frac{q-1}{1-p}\ln\frac{1}{\delta}}\rceil$ with probability at least $1-4\delta$.

Proof: Let us say that a cell (x, y) is even if $x+y$ is even and odd if $x+y$ is odd. Our row and column indices start with 0. For example, the upper right cell is considered as an even cell. At any point away from the right boundary, from an even cell one moves down (respectively, right) when the first bit (respectively, second bit) flips, and vice versa for the odd cells. For the theorem we consider the case $p \ge 1/2$ (the case of $p < 1/2$ follows by symmetry, with $1-p$ replaced by p in corresponding term in the deficiency). Consider the first $2(q-1) - 4\lceil\sqrt{\frac{q-1}{1-p}\ln\frac{1}{\delta}}\rceil$ transitions, which are necessarily split as evenly as possible between the even and odd cells; we show that with probability $1-4\delta$ we do not reach a boundary before $2(q-1) - 4\sqrt{\frac{q-1}{1-p}\ln\frac{1}{\delta}}$ transitions complete when starting at (0,0). We ignore floors and ceilings for ease of notation, and similarly we say there are $q - 1 - 2\sqrt{\frac{q-1}{1-p}\ln\frac{1}{\delta}}$ moves each at even and odd cells, as rounding and differences by 1 are asymptotically unimportant. For the moves in even cells, the expected number of down moves is $p(q-1-2\sqrt{\frac{q-1}{1-p}\ln\frac{1}{\delta}})$, and similarly, for odd cells, the expected number of down moves is $(1-p)(q-1-2\sqrt{\frac{q-1}{1-p}\ln\frac{1}{\delta}})$. Let M_0 be the number of down moves from even cells and M_1 be the number of down moves from odd cells. As each move is an independent trial (as long as neither boundary is hit), it follows from standard Chernoff bounds[1] [20, Theorem 4.4, Part 2] that for q sufficiently large

$$\Pr\left(M_0 \ge p\left(q-1-2\sqrt{\frac{q-1}{1-p}\ln\frac{1}{\delta}}\right) \cdot\left(1+2((1-p)(q-1))^{-1/2}\sqrt{\ln\frac{1}{\delta}}\right)\right) \le \delta$$

which implies

$$\Pr(M_0 \ge p(q-1)) \le \delta.$$

Similarly

$$\Pr\left(M_1 \ge (1-p)\left(q-1-2\sqrt{\frac{q-1}{1-p}\ln\frac{1}{\delta}}\right) \cdot\left(1+2((1-p)(q-1))^{-1/2}\sqrt{\ln\frac{1}{\delta}}\right)\right) \le \delta$$

so

$$\Pr(M_1 \ge (1-p)(q-1)) \le \delta$$

and we have similar results for right moves. It follows that the number of down (or right) moves in the first $2q-2-4\sqrt{\frac{q-1}{1-p}\ln\frac{1}{\delta}}$

[1]Specifically, we use the simple Chernoff bound that says for the sum X of independent trials with overall mean μ, for any γ with $0 < \gamma < 1$, $\Pr(X \ge (1+\gamma)\mu) \le e^{-\mu\gamma^2/3}$.

transitions is less than $q-1$ with probability at least $1-4\delta$ as claimed. □

As should be clear from the proof, one can trade off the magnitude of the deficiency with the success probability; for example, we can also achieve a success probability of $1-1/\mathrm{poly}(q)$ without hitting a boundary in $2q-O(\sqrt{q\log q})$ transitions, giving a deficiency that is $O(\sqrt{q\log q})$. More generally, one simply has $o(q)$ deficiency with high probability in q.

It should be noted that our assumption regarding the underlying probabilistic model is particularly important. If the two bits alternate flipping, then the number of transitions before a reset will be much less than $2(q-1)$, as a boundary will be reached in only $q-1$ transitions.

B. The Case of $k=2, \ell=2, n>2$

We now consider the case when $n>2$. Here we provide multiple different approaches. While this may seem excessive, we recall that we view these results as building blocks for future work, and our different approaches have subtly different properties.

For our first approach, we first note that for even values of n, we can simply successively glue together the $n=2$ result multiple times. That is, we split the n dimensions of the cell state vector into $n/2$ *subblocks* each of two dimensions, and use a 2DGC code for each pair of dimensions. The current variable value vector is given by taking the exclusive-or (XOR) of the corresponding variable vector of all of the subblocks. With this setup, one first moves for $2(q-1)-o(q)$ transitions in the first two dimensions, until one hits a point where a reset would occur, and then one makes no further transitions in these two dimensions; then $2(q-1)-o(q)$ transitions are done in the next two dimensions, and so on, to obtain a total of $n(q-1)-o(nq)$ transitions with high probability. One must take some care to be clear what is meant with high probability in this case, however. When n is large enough (exponential in q), almost surely for some pair of dimensions a boundary will be reached after only $q-1$ transitions and not $2(q-1)-o(q)$ transitions. Of course such occurrences should be rare, and not affect the final result significantly. As the intuition is clear, and as we will follow with what will in most cases be a tighter result, we simply sketch the proof. For each of the $n/2$ pairs of dimensions, let X_i be the random variable corresponding to the deficiency of the ith pair; that is, the ith pair performs $2(q-1)-X_i$ transitions before hitting the boundary. It follows straightforwardly from Theorem 1 that the deficiency for each X_i has an expectation of $O(\sqrt{q})$ and has geometrically decreasing tails. It follows that the sum of the X_i has expectation $O(n\sqrt{q})$, and again using standard Chernoff bounds on the sum of the X_i the total deficiency is $o(nq)$ with high probability. (Alternatively, one could approximate each X_i by a Gaussian random variable for large q to obtain a similar result.)

To handle odd n as well, it suffices to demonstrate a code for the case $n=3$ that allows $3(q-1)+o(q)$ moves with high probability. One could then deal with $n=2m+3$ dimensions by handling the first $2m$ dimensions as above and the last three with this code. The construction for $n=3$, interesting in its own right, is also based on Gray codes, although some care must be taken to handle the third dimension properly. We again represent the code pictorially, with the value in the ith row and jth column from the kth square representing the value vector under D for the cell state vector (i,j,k). An example for $q=4$ is given in Fig. 3, and for $q>4$ the code is again obtained by cycling through the Gray code values repeatedly in each two-dimensional square. Note that the cell state vectors for a given i and j are the same for all even values of k, and the same for all odd values of k. As an example, the decoded value for cell state (2,1,1) is 11, as underlined in the table. This gives us a 3-dimensional Gray Code (3DGC). For the 3DGC, there is some ambiguity, as a single transition of a value can lead to multiple cells of distance 1 from the current cell in the 3DGC. We resolve this ambiguity by having our rewriting function, from each cell, move to the closest available cell that decodes to the proper new value, with priority toward moving *up* in the third dimension. That is, from cell state (x,y,z), when possible our move increases the z value by 1. (Note *up* and *down* are not opposites here.)

	y =	0	1	2	3
z=0	x=0	00	01	11	10
	x=1	10	00	01	11
	x=2	11	10	00	01
	x=3	01	11	10	00
z=1	y =	0	1	2	3
	x=0	10	00	01	11
	x=1	11	10	00	01
	x=2	01	<u>11</u>	10	00
	x=3	00	01	11	10
z=2	y =	0	1	2	3
	x=0	00	01	11	10
	x=1	10	00	01	11
	x=2	11	10	00	01
	x=3	01	11	10	00
z=3	y =	0	1	2	3
	x=0	10	00	01	11
	x=1	11	10	00	01
	x=2	01	11	10	00
	x=3	00	01	11	10

Fig. 3. The code, or the 3DGC, for $k=2, \ell=2, n=3, q=4$. The upper left corner in the first square represents the decoded value vector for cell (0,0,0); the lower right corner in the last square represents the decoded value vector for cell (3,3,3). Transitions are greedy, breaking ties by increasing the z coordinate.

To avoid using the same heavy notation of Theorem 1, we will prove a slightly weaker result for the 3DGC. In particular, we will only prove that the deficiency of 3DGC is $o(q)$ with high probability, without quantifying in detail the relation between a bound on the deficiency and its probability.

Theorem 2: For any p, the number of transitions before a reset for the 3DGC is $3(q-1)-o(q)$ with high probability (in q).

Proof: For a cell (x,y,z), we say that it is z-even if the z coordinate is even and that it is xy-even if the sum $x+y$ is even, and similarly for z-odd and xy-odd. With the given code, the moves can be classified according to four cell types (away from the boundaries):

- xy-even and z-even: move up with probability p and right with probability $1-p$;

- xy-even and z-odd: move up with probability p and down with probability $1-p$;
- xy-odd and z-even: move up with probability $1-p$ and right with probability p;
- xy-odd and z-odd: move up with probability $1-p$ and down with probability p.

We first note that, away from the top boundary $z = q-1$, the probability that we fail to move up in at least one of the next two moves it at most $p(1-p) \leq 1/4$. It follows easily that, for any constant $\gamma > 0$, as long as we don't hit one of the boundaries in the x or y dimension, the number of up moves in the first $3(q-1) - 3q^{1/2+\gamma}$ transitions after any reset will be $q-1$ with high probability for sufficiently large q.

We now want to show that the number of down and right moves until we reach $z = q-1$ are essentially split equally. This ensures we do not hit another boundary before $z = q-1$, and since when we reach $z = q-1$ we will be in the same setting as the 2DGC, it also implies that $2q - 2q^{1/2+\gamma}$ down and right moves are performed with high probability.

There is a minor complication in that the number of moves spent on a level $z = a$ before moving up to $z = a+1$ depends on whether we start at that level on an xy-even or xy-odd cell, which in turn depends on the value of a and the starting point after the last reset. However, the probability of starting at an xy-even cell after moving up quickly converges to it equilibrium value (which is p) after a small number of levels, and so we may ignore this complication, as the difference is absorbed into the $o(q)$ term. Given that we start at an xy-even cell, and are away from any of the boundaries, the probability that we stay at the same level z for k moves (always moving right when starting at a z-even cell, and down for a z-odd cell) is the same for every value of z, and is further dominated by a geometric distribution. The same holds for xy-odd cells. Standard applications of Chernoff bounds therefore again give with high probability the deviation between down and right moves over the first $3(q-1) - 3q^{1/2+\gamma}$ moves is $o(q)$ with high probability, and the result follows. □

We emphasize that for any specific value of q, Theorem 2 does not state that the 3DGC is the optimal code. Indeed, Gray codes can be layered in different ways to obtain possible codes, and the parameters in any given circumstance might determine which performs best.

Looking again now at the result for the 2DGC code, it appears that we should be able to do significantly better than an overall deficiency of $O(n\sqrt{q})$. We have pessimistically bounded performance by assuming that once we reach a boundary in each two-dimensional subblock of the code, we essentially never move in that subblock again. One can prove an expected deficiency of only $O(\sqrt{q})$ by allowing subsequent greedy moves along the boundary of a subblock whenever such a move only changes a cell value by 1. The proof of this is somewhat more difficult; however, we can obtain the same result more easily taking advantage of a technique motivated by the worst-case analysis of [24]. For convenience we again suppose that n is even (if n is odd, we can use a final subblock of 3 cells without changing the asymptotics). We split the n dimensions into $n/2$ subblocks, with the cell values for each pair of dimensions corresponding to values for a 2DGC code, and the value of the two variable bits being the XOR of the corresponding decoded values. For every subblock of two dimensions, we say the subblock is *empty* if both cell values are 0, *full* if both values are $q-1$, and *active* otherwise. Taking advantage of the ideas of [24] leads to the following theorem (which again we will prove in a form weaker than the one of Theorem 1).

Theorem 3: Let n be even, $\gamma > 0$ be a fixed constant, and $k = \ell = 2$. Then there is a floating code that at any point in time has at most two active subblocks, and under the model where at each step the next bit to flip is the first with probability p, the final deficiency is $O(q^{1/2+\gamma})$ with high probability in q.

Proof: Following [24], we think of our $n/2$ subblocks as being laid out from left to right. Initially, consider the leftmost subblock of two dimensions, and the rightmost subblock of two dimensions. Each subblock will utilize a 2DGC code, but we change the cell state for the leftmost subblock only when the first bit flips, and the cell state for the rightmost subblock only when the second bit flips. As before, the variable values are given by taking the XOR of the value corresponding to all blocks (or equivalently to the XOR of all full and active blocks). Notice that, for each subblock, using the 2DGC code we simply walk along the diagonal until the subblock becomes full. Once the leftmost subblock is full, on the next flip of the first bit, the second subblock from the left becomes active and tracks the flips of the first bit, and similarly for the rightmost subblock. We call this the 2DGC-n code.

Clearly there are at most two active subblocks at any point in time. As long as there is an empty subblock between two active subblocks, there is no ambiguity about which subblocks are to be used at any step on a flip of each bit. Indeed, the only concern is when, at some point, we are left with only a single subblock. At the first step when we are left with just one subblock, we start either on or adjacent to a cell state on the diagonal of our 2DGC code. It therefore follows from the same argument as Theorem 1 that when we terminate the final deficiency is $O(q^{1/2+\gamma})$ with high probability in q. □

In [24] similar codes were used as a building block for larger codes, that is with a larger number of bits k. We expect our codes can be used in a similar fashion, but this is left as an issue for future work.

IV. The Case of $k = 3, \ell = 2, n = 2$

Continuing our exploration of small cases and the use of Gray codes, we here show how to utilize Gray codes to obtain a code that performs well for the case of $k = 3, \ell = 2, n = 2$. Our model for the three bits of data is that at each time step the first is the next to change with probability p_1, the second with probability p_2, and the third with probability $p_3 = 1 - p_1 - p_2$. We again consider the asymptotic behavior as q grows large, where our decoding function for $q = 8$ is given by Fig. 4 and for larger q is obtained by cycling in each dimension. Also, the transition function (described in more detail below) is the standard move to the nearest suitable cell. Notice that now, by necessity, we can not arrange to move only to an adjacent cell on a transition; with our Gray code design, some transitions require

000	100	101	001	011	111	110	010
010	000	100	101	001	011	111	110
110	010	000	100	101	001	011	111
111	110	010	000	100	101	001	011
011	111	110	010	000	100	101	001
001	011	111	110	010	000	100	101
101	001	011	111	110	010	000	100
100	101	001	011	111	110	010	000

Fig. 4. The basic building block for the 2DGC-3, where $k = 3$ and $l = 2$.

W	X	Y	Z
Z	W	X	Y
Y	Z	W	X
X	Y	Z	W.

Fig. 5. An alternative view of the basic building block for the 2DGC-3.

moving three cells down or to the right. For convenience, we refer to this code as 2DGC-3. By classifying these transitions, we obtain the following result:

Theorem 4: The number of transitions before a reset for the 2DGC-3 is $2(q-1)/(2-p_1) - o(q)$ with high probability (in q).

Proof: Upon examination, away from the boundaries, there are four types of cell states that for convenience we label as W, X, Y, and Z.

- W: move right 1 when first bit flips, down 1 when second bit flips, right 3 when third bit flips;
- X: move down 1 when first bit flips, down 3 when second bit flips, right 1 when third bit flips;
- Y: move right 1 when first bit flips, right 3 when second bit flips, down 1 when third bit flips;
- Z: move down 1 when first bit flips, right 1 when second bit flips, down 3 when third bit flips.

The 2DGC-3 code consists of repeated blocks of the form given in Fig. 5.

Consider W, X, Y, and Z as states of a Markov chain. Then the Markov process on how value bits change induces the following Markov chain on these four states:

- W: moves to X with probability p_1, moves to Z with probability $(1-p_1)$;
- X: moves to W with probability p_1, moves to Y with probability $(1-p_1)$;
- Y: moves to Z with probability p_1, moves to X with probability $(1-p_1)$;
- Z: moves to Y with probability p_1, moves to W with probability $(1-p_1)$.

A straightforward analysis gives that, for any constant c, over the first $c(q-1)-q^{2/3}$ transitions after a reset, regardless of the initial starting point, with high probability at most $c(q-1)/4+o(q^{2/3})$ of the steps are spent in each of the states W, X, Y and Z as long as no boundary is reached. Following the same reasoning as in Theorem 1, the expected number of total spaces moved down or to the right in the first $c(q-1)-q^{2/3}$ transitions is $c(2-p_1)(q-1)/2-o(q)$ with high probability (in q). Choosing $c = 2/(2-p_1)$, we obtain that we avoid the boundary and a reset for $2(q-1)/(2-p_1)-o(q)$ transitions with high probability.□

Similar analyses can be performed for larger values of k; however, we do not have a proof that such codes are asymptotically optimal for $k > 2$, as we do not at this point have an upper bound.

An interesting consideration brought on by this result is the idea that we could want multiple codes; here, we could have three similar but distinct codes, with the ith code taking $2(q-1)/(2-p_i) - o(q)$ transitions with high probability. We could then dynamically choose the best code based on recent behavior, so that if the relative frequency of each bit flipping changes, our code could, after a reset, conceivably change with it.

V. Considering Random Codes

We believe some additional insight can be gained by considering the behavior of random codes in this setting, which do not appear to have been considered in previous work. By a random code here, we mean that each element in the range of the decoding function D is chosen independently and uniformly at random from $\{0, 1, \ldots, \ell-1\}^k$. It turns out that results are easier to obtain in the more general model where, at each step, the variable vector can change entirely, not just in a single coordinate. In this model, there is a request sequence $(r_1, r_2, \ldots, r_z)$, where each request $r_i \in \{0, 1, \ldots, \ell-1\}^k$ is a value representing the ith rewrite of the *entire* variable vector. The cell state vector must be updated accordingly. Note that here one request would correspond to a series of up to k requests in the model where a single value changes, so our previous results do not provide good performance (we would lose up to a factor of k).

The intuition behind our results is that, if n is sufficiently large (greater than ℓ^k), for each request we should be able to increase just one coordinate in the cell state vector by 1 to achieve a cell state vector that decodes properly. This defines the rewriting function associated with the random decoding function. Here, we use this intuition to show the existence of a code with deficiency at most $O(q\ell^k \log n)$ for *any* input sequence. While this may not be optimal (we discuss our relation to [6], [17] below), the proof, which utilizes the Lovász Local Lemma, may be useful for future work. We also show that, for any fixed input sequence, the expected deficiency is only $O(q\ell^k)$ using a straightforward algorithm, which is asymptotically optimal in the sense that it is $o(n)$. With a more complicated algorithm, we can reduce the expected deficiency further to $O(\ell^k)$.

We first recall the Lovász Local Lemma [20].

Theorem 5 (Lovász Local Lemma): Let $\mathcal{E}$ be a set of *bad* events over some probability space. Also, let p and Δ be two numbers satisfying $4p\Delta \leq 1$, such that for each $E_i \in \mathcal{E}$,

- $\Pr[E_i] \leq p$,
- E_i is mutually independent of a set of all but at most Δ of the events in $\mathcal{E}$.

Then with positive probability none of the events in $\mathcal{E}$ occurs.

Using this, we can show the existence of a random code as aforedescribed.

Theorem 6: There exists a decoding function D that allows the algorithm to serve without resetting each sequence of value

requests $(r_1, \ldots, r_t)$ with $r_i \in \{0, 1, \ldots, \ell-1\}^k$ and $t = n(q-1) - 3q\ell^k \log(2n)$.

Proof: Consider the cell state vector $\bar{c} = (c_1, c_2, \ldots, c_n)$. We define the weight of $\bar{c}$ as $w(\bar{c}) = \sum_{i=1}^{n} c_i$. We also say that $\bar{c}' = (c'_1, c'_2, \ldots, c'_n)$ is an out-neighbor of $\bar{c}$ if there exists some i such that $c'_i = c_i + 1$ and $\forall j \neq i, c'_j = c_j$. If $\bar{c}'$ is an out-neighbor of $\bar{c}$, then $\bar{c}$ is an in-neighbor of $\bar{c}'$.

We show the existence of a decoding function D such that for each cell state vector $\bar{c}$ with $w(\bar{c}) \leq n(q-1) - 3q\ell^k \log(2n)$ and for each value $r \in \{0, 1, \ldots, \ell-1\}^k$, $\bar{c}$ has some out-neighbor containing value vector r. This immediately implies the claim.

We use the Lovász Local Lemma. The probability space is the natural one: each value $D(\bar{c})$ is chosen independently and uniformly at random from $\{0, \ldots, \ell-1\}^k$. For each $\bar{c}$, with $w(\bar{c}) \leq n(q-1) - 3q\ell^k \log(2n)$, we define the bad event $B_{\bar{c}}$ to be that there exists some value vector r such that no out-neighbor of $\bar{c}$ decodes to r. To bound the dependency between bad events, note that each bad event $B_{\bar{c}}$ depends only on the value assigned to each out-neighbor of $\bar{c}$, and hence $B_{\bar{c}}$ shares a dependency with no more than $\Delta = n^2$ other bad events.

The probability of a bad event $B_{\bar{c}}$ can be bounded as follows. When $w(\bar{c}) \leq n(q-1) - 3q\ell^k \log(2n)$, $\bar{c}$ has at least $3\ell^k \log(2n)$ positions with values less than q. Each position corresponds to an out-neighbor of $\bar{c}$. Now fix any value vector r. The probability that the at least $3\ell^k \log(2n)$ different out-neighbors fail to decode to r is at most

$$\left(1 - \frac{1}{\ell^k}\right)^{3\ell^k \log(2n)} \leq \frac{1}{8n^3}.$$

There are ℓ^k different value vectors, and ℓ^k must be less than n for the theorem to be nontrivial. By a union bound, the probability that some value vector cannot be obtained from the decoding of an out-neighbor of $\bar{c}$ is upper-bounded by $p = \frac{1}{8n^2}$.

As $4p\Delta \leq 1$, the Lovász Local Lemma applies, and there exists some decoding function with the desired property. □

We note the logarithmic dependence on n in the argument arises from the value of Δ, and the dependence on ℓ^k arises because each value of the decoding function is chosen randomly over all ℓ^k possible values. It seems difficult to improve these values with this argument, even in the restricted model where only one variable value (and not the entire vector) is allowed to change at each step. However, analysis of worst-case codes has improved recently, giving better bounds. In [6] a code with worst-case deficiency $O(q\ell^k \log^2(\ell^k))$ is given (only the case $\ell = 2$ is considered there, but the result given generalizes easily) and further improvements in [17] reduce the worst-case deficiency to $O(\max\{q, \log \ell^k\}\ell^k \log(\ell^k))$. The best of these results dominate ours when n is large; one could start with the code of [17] and then switch to the code guaranteed by Theorem 6 when n is small to have the best of both results, but this does not change the asymptotics.

We can also naturally improve the bound by considering the expected deficiency on a *fixed sequence* for the natural greedy decoding algorithm, instead of the worst case over all sequences.

Theorem 7: Fix a request sequence $(r_1, \ldots, r_{n(q-1)})$, and choose a random code. The expected deficiency (over the random choice of code) on that request sequence is at most $\ell^k(q-1)$.

Proof: Again, we assume a greedy algorithm for defining the rewriting function: if increasing a single coordinate by one will lead to a cell state vector that decodes to the proper variable value, make such a move, choosing randomly from all possible such moves if there is more than one. However, in this case, if no such move exists, we simply choose a coordinate independently and uniformly at random that does not have value $q-1$, increase it by one, and continue to look greedily for a single increase in a single coordinate that leads to a suitable cell state vector. (If none is found by repeating this greedy method, a reset eventually occurs when the block is full, all cells having value $q-1$.) Let X_i be a 0/1 random variable that takes the value 1 if the request is handled successfully when the weight of the cell state vector is i. Using the principle of deferred decisions [20], we can randomly assign the decoding value for the cell state vectors of weight i only after we have reached the point in the request sequence where the cell state vector has weight $i-1$. Then we see that the X_i are independent, and the number of out-neighbors when the weight of the cell state vector is i is at least $\lceil n - \frac{i}{q-1} \rceil$. Hence the expected number of successful moves until the block is full is at least

$$\begin{aligned}
&\sum_{i=0}^{n(q-1)-1} \left(1 - \left(1 - \frac{1}{\ell^k}\right)^{\lceil n - \frac{i}{q-1} \rceil}\right) \\
&= n(q-1) - \sum_{i=0}^{n(q-1)-1} \left(1 - \frac{1}{\ell^k}\right)^{\lceil n - \frac{i}{q-1} \rceil} \\
&= n(q-1) - (q-1)\sum_{j=1}^{n} \left(1 - \frac{1}{\ell^k}\right)^{j} \\
&\geq n(q-1) - \ell^k(q-1).
\end{aligned}$$

This yields the theorem. □

The proof of Theorem 7 pessimistically assumes a worst-case bound on the number of out-neighbors as a function of the weight. It seems that if we had a finer analysis of the number of out-neighbors as a function of weight for the process, we would improve this bound.

Here we consider another improvement by more carefully choosing moves. If there are several possible cells where increasing a single coordinate by one would lead to the proper variable value, the natural strategy is to choose to increase the value of the cell with the smallest current value, instead of randomly. Surprisingly, assuming that n is sufficiently large, we can give a very detailed analysis of this approach, showing that the expected deficiency is at most $\ell^k + o(1)$. Call this encoding algorithm Least.

Theorem 8: The Least algorithm with a random code has expected deficiency $\ell^k + o(1)$ on any fixed input sequence, where the $o(1)$ term is vanishing with n, as long as $\ell^k \log(nq) \in o(\sqrt{n})$.

Proof: We divide the process in phases: the first phase will run through requests $r_1, \ldots, r_n$, while the ith phase $(1 \leq i \leq q-1)$ will run through requests $r_{n(i-1)+1}, \ldots, r_{ni}$.

TABLE I
AVERAGE LONG-TERM COST (RESETS/MOVES) OF VARIOUS FLOATING CODES AS p VARIES

Scheme	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
2DWC ($q=4$)	0.2088	0.2104	0.2134	0.2175	0.2222	0.2273	0.2324	0.2379	0.2438
2DGC ($q=4$)	0.2119	0.2146	0.2165	0.2176	0.2180	0.2175	0.2164	0.2146	0.2120
2DGC+ ($q=4$)	0.1763	0.1831	0.1874	0.1897	0.1905	0.1898	0.1874	0.1831	0.1763
2DWC ($q=8$)	0.0919	0.0926	0.0935	0.0944	0.0952	0.0962	0.0971	0.0980	0.1000
2DGC ($q=8$)	0.0797	0.0811	0.0820	0.0825	0.0827	0.0826	0.0820	0.0811	0.0797
2DGC+ ($q=8$)	0.0753	0.0771	0.0780	0.0785	0.0787	0.0786	0.0780	0.0771	0.0753
2DWC ($q=12$)	0.0592	0.0595	0.0599	0.0602	0.0606	0.0610	0.0613	0.0617	0.0625
2DGC ($q=12$)	0.0491	0.0499	0.0504	0.0506	0.0507	0.0506	0.0504	0.0499	0.0491
2DGC+ ($q=12$)	0.0476	0.0484	0.0489	0.0492	0.0492	0.0491	0.0489	0.0484	0.0476
3DWC ($q=4$)	0.1460	0.1466	0.1473	0.1480	0.1481	0.1479	0.1474	0.1466	0.1460
3DGC ($q=4$)	0.1287	0.1310	0.1326	0.1334	0.1333	0.1322	0.1300	0.1273	0.1243
3DWC ($q=8$)	0.0596	0.0601	0.0607	0.0611	0.0615	0.0619	0.0622	0.0625	0.0629
3DGC ($q=8$)	0.0514	0.0521	0.0526	0.0528	0.0528	0.0525	0.0521	0.0514	0.0505
3DWC ($q=12$)	0.0374	0.0378	0.0382	0.0385	0.0388	0.0391	0.0394	0.0397	0.0400
3DGC ($q=12$)	0.0321	0.0324	0.0327	0.0328	0.0328	0.0327	0.0325	0.0322	0.0317

We say that phase $i < q-1$ *ends well* if after request r_{ni}, there exists $c_i \leq T = 4\ell^k \log(nq)$, such that

- c_i cells have value $i-1$;
- c_i cells have value $i+1$; and
- $n-2c_i$ cells have value i.

We show that, with probability at least $1-O((nq)^{-2})$, phase 1 ends well with all requests requiring only a single cell increment. Also, for each $i = 2, \ldots, q-2$, we show that if phase $i-1$ ends well, then with probability at least $1-O((nq)^{-2})$ phase i also ends well with all requests requiring only a single cell increment. Finally, we will show that if phase $q-2$ ends well, then the expected deficiency from phase $q-1$ is at most $\ell^k + o(1)$, proving the statement.

Suppose that, for some $2 \leq i \leq q-2$, phase $i-1$ ended well. Then, at the beginning of phase i, the set C_{i-2} of cells of value $i-2$ has cardinality $|C_{i-2}| = c_{i-1} \leq T$; also, $|C_{i-1}| = n-2c_{i-1}$ and $|C_i| = c_{i-1}$.

First consider the cells in $|C_{i-2}|$ at the beginning of the phase. If at least one such cell exists, the probability that no such cell will be incremented after T requests is at most $(1-\frac{1}{\ell^k})^T \leq \frac{1}{n^4q^4}$. Thus after at most $c_{i-1} \cdot T \leq T^2 = o(n)$ many requests, C_{i-2} will be empty with probability at least $1-O(\frac{1}{n^3q^3})$.

Also, as long as $|C_{i-2}| + |C_{i-1}| \geq T$, the probability that a request cannot be served by incrementing a cell of value at most i is upper-bounded by

$$\left(1-\frac{1}{\ell^k}\right)^T = \left(1-\frac{1}{\ell^k}\right)^{4\ell^k \log(nq)} \leq \frac{1}{n^4q^4}.$$

Thus, with probability at least $1-O(\frac{1}{n^3q^3})$, after $n-T$ requests of phase i: (a) C_{i-2} will be empty; (b) C_{i-1} will have size T (it had size $n-2k, c_{i-1}$ elements got to it from C_{i-2}, and $n-T-c_{i-1}$ elements of C_{i-1} moved to C_i); (c) C_i will have size $n-T$; and (d) C_{i+1} will be empty.

In the last T requests of the phase, again with probability at least $1-O(\frac{1}{n^3q^3})$, each element will be served by either a move from C_{i-1} to C_i or from C_i to C_{i+1}. Say that the former case happens $t \geq 0$ times. Since $t \leq T$, at the end of the phase $|C_{i-1}| = |C_{i+1}| = T-t \leq T$, while $|C_i| = n-2(T-t)$.

The same reasoning, with small modifications (that is, $|C_{i-2}|$ set to 0) also applies to phase 1. Hence with high probability (at least $1-O(\frac{1}{n^2q^2})$) all phases up to $q-2$ end well with all requests requiring a single cell increment. As the maximum possible deficiency is $n(q-1)$, the bad events have only a $o(1)$ influence on the expectation.

So it only remains to be proved that the expected deficiency of phase $q-1$ is at most $\ell^k + o(1)$ if phase $q-2$ ended well. In this case at the beginning of phase $q-1$ we have $|C_{q-3}| = |C_{q-1}| \leq T$. Also, with high probability, after $c_{i-3}T \leq T^2$ many requests C_{q-3} will become empty, leaving $|C_{q-2}| = a$ and $|C_{q-1}| = b$, for some a, b such that $a+b = n$. Again, with high probability, up to this point all requests take a single cell increment. From that point the expected additional deficiency of phase $q-1$ is bounded by

$$\sum_{j=0}^{a}\left(1-\frac{1}{\ell^k}\right)^{a-j} = \sum_{j=0}^{a}\left(1-\frac{1}{\ell^k}\right)^{j} \leq \sum_{j=0}^{\infty}\left(1-\frac{1}{\ell^k}\right)^{j} = \ell^k.$$

The result follows. □

Finally, it is also worth noting that there is a simple $\Omega(\min(\ell^k, n)q)$ lower bound for the deficiency in the worst case where the request sequence is chosen by an adversary who is given the code. This is a simple generalization of [13, Theorem 2], so we do not repeat it here.

VI. SIMULATIONS AND COMPARISONS

A. 2DGC and Related Codes When $n = 2$

We present some results based on simulating the performance of our codes. In Table I, we compare three different codes for the setting where $k = \ell = n = 2$. First, we consider the 2-dimensional Gray Code (2DGC) previously described. We also consider a modified version, 2DGC+, described below. Finally, we consider the worst-case optimal code (2DWC), from [13]. The resulting average long-term costs, corresponding to (1), are obtained by running the appropriate Markov chain for 100 000 000 steps. (While we could have instead determined the values by explicitly computing the equilibrium distribution for each chain, we found it useful to develop a simulator in the process of our work.) We vary p across a range of values, and present the corresponding costs (to four decimal places). We also similarly consider the 3-dimensional Gray Code (3DGC) and the worst-case optimal code for three dimensions (3DWC).

A simple but useful improvement on the 2DGC, especially when q is small, is to change the lower-right corner to 11 from

00	01	11	10
10	00	01	11
11	10	00	01
01	11	10	11

Fig. 6. The 2DGC+. Changing the lower right corner gives a slight improvement.

00, as in Fig. 6. This is what we call 2DGC+. This change improves the average cost because if the process is at 00 on the lower-right corner, the next transition will cause a reset to the upper leftmost 01 or 10 position. However, the process would transition to the same position when starting from the 00 in the upper left corner. Hence, when the process reaches the cell 00 in the lower right hand corner, it saves nothing over resetting to the upper left corner, since the next move will lead to the same position. The cost is therefore reduced by instead using the cell to hold value pair 11, which can usefully prevent a reset for one further move on some occasions.

As can be seen from Table I, with these parameters the 2DGC code performs better than the worst-case code, except for some cases where $q = 4$. This is perhaps not surprising, in that the worst-case code was designed with a different consideration in mind. Our main point, however, it that it is interesting that a single code outperforms this code over the entire range of p. The 2DGC+ variation performs even slightly better than the 2DGC, although the gap quickly vanishes with q, as one would expect. The additive gap between the worst-case code and both the 2DGC and 2DGC+ also declines with q, but the multiplicative gap actually appears to increase. That is, assuming that the lifetime is dominated by the time between reset operations and that our model is suitable, the percentage increase in lifetime by using 2DGC or 2DGC+ as the underlying code is increasing with q.

Similar results are obtained with the 3DGC, also presented in Table I. Again, we see significant improvements over the worst case code across the entire range of p. Hence, while our results for these codes are asymptotic, and they may in fact not be optimal, the principle behind their design yields demonstrably better performance over worst-case codes given our assumptions.

Finally, we note that the 2DGC and 2DGC+ codes are symmetric and obtain the same values for p and $1-p$; the differences in our table arise because we are simulating performance. The 2DWC, 3DWC, and 3DGC codes are not symmetric.

B. Extensions to Larger n

We recall that in Section III-B we described multiple ways to extend the 2DGC approach to larger values of n, again with $k = \ell = 2$. The first (Method A) was simply to glue together multiple 2DGC codes, starting a new pair of cells whenever a boundary is reached. As expected, simulations show that the average deficiency in this case grows like $O(n\sqrt{q})$, as shown by the sample data in Table II. Here the averages are taken over 100 trials for $p = 0.5$; other values of p have similar behavior. Our second approach (Method B) also glues together multiple 2DGC codes but moves single steps along the boundary of old cells when possible. Our final approach (Method C) utilizes 2DGC codes but uses the left-right methodology of [24]. Methods B and C are compared in Table III. Both obtain excellent performance, nearly the same but with Method C performing slightly better.

TABLE II
AVERAGE DEFICIENCY OF THE TRIVIAL APPROACH (METHOD A) WHEN $p = 0.5$, WHICH GROWS ROUGHLY PROPORTIONALLY TO $n\sqrt{q}$

n	q	A
256	4	150.5
256	16	449.31
256	64	1024.55
256	256	2187.88
1024	4	607.54
1024	16	1780.35
1024	64	4094.06
1024	256	8715.3
4096	4	2427.59
4096	16	7113.57
4096	64	16380.77
4096	256	34720.27

TABLE III
AVERAGE DEFICIENCY FOR METHODS B AND C, WHICH GROW ROUGHLY PROPORTIONALLY TO $\sqrt{q}$

n	q	p	B	C
256	4	0.1	0.85	0.58
256	16	0.1	1.67	1.24
256	64	0.1	4.43	2.88
256	256	0.1	10.02	4.33
1024	4	0.1	0.74	0.66
1024	16	0.1	1.86	1.46
1024	64	0.1	4.28	2.98
1024	256	0.1	9.08	6.92
4096	4	0.1	0.74	0.73
4096	16	0.1	1.62	1.24
4096	64	0.1	4.68	3.21
4096	256	0.1	8.99	6.48
16384	4	0.1	0.92	0.59
16384	16	0.1	1.76	1.31
16384	64	0.1	4.88	3.32
16384	256	0.1	10.79	6.76
65536	4	0.1	0.71	0.62
65536	16	0.1	1.66	1.35
65536	64	0.1	4.21	3.00
65536	256	0.1	9.37	5.78
256	4	0.5	1.02	0.85
256	16	0.5	3.58	2.02
256	64	0.5	7.5	5.49
256	256	0.5	19.67	10.91
1024	4	0.5	1.06	0.68
1024	16	0.5	3.44	2.06
1024	64	0.5	7.89	5.54
1024	256	0.5	17.76	11.51
4096	4	0.5	1.01	0.78
4096	16	0.5	3.43	2.21
4096	64	0.5	7.8	5.33
4096	256	0.5	17.71	11.34
16384	4	0.5	1.16	0.69
16384	16	0.5	3.11	2.60
16384	64	0.5	7.97	4.59
16384	256	0.5	18.72	10.81
65536	4	0.5	1.12	0.77
65536	16	0.5	3.07	2.38
65536	64	0.5	7.79	4.53
65536	256	0.5	16.62	10.16

C. Random Codes

For our simulations of random codes, we study the two schemes described in Section V. In one scheme, we choose

TABLE IV
AVERAGE DEFICIENCY FOR SIMPLE AND LEAST STRATEGIES USING RANDOM CODES

n	ℓ^k	q	Simple	Least
256	16	4	20.91	14.63
256	64	4	106.40	73.85
256	256	4	402.81	364.78
256	16	16	36.76	15.05
256	64	16	256.09	129.34
256	256	16	1685.70	1499.01
256	16	64	65.88	14.68
256	64	64	659.46	344.90
256	256	64	6485.42	6001.73
256	16	256	126.67	15.47
256	64	256	1894.61	1217.85
256	256	256	25076.50	24083.52
1024	16	4	19.64	15.20
1024	64	4	86.24	63.62
1024	256	4	428.54	299.25
1024	16	16	32.26	15.08
1024	64	16	152.15	63.99
1024	256	16	1049.48	527.35
1024	16	64	54.04	15.26
1024	64	64	275.45	62.34
1024	256	64	2673.70	1412.89
1024	16	256	97.85	14.18
1024	64	256	523.13	62.33
1024	256	256	7698.46	4992.36
4096	16	4	18.17	14.95
4096	64	4	80.10	62.82
4096	256	4	350.86	255.17
4096	16	16	28.54	14.90
4096	64	16	130.51	63.74
4096	256	16	615.75	256.58
4096	16	64	47.92	15.63
4096	64	64	228.63	63.08
4096	256	64	1116.98	254.79
4096	16	256	85.63	15.09
4096	64	256	422.87	63.30
4096	256	256	2121.23	254.52

randomly a move that increases a single coordinate by 1 if one is available, and otherwise we increment a coordinate independently and uniformly at random and continue; call this algorithm Simple. We also consider the scheme where we choose the move that increases the least loaded coordinate by 1 if there exists a move that puts us in the right state, and otherwise we increment the least loaded coordinate and continue; call this algorithm Least. Recall that at any step the variable vector can change completely, and there are ℓ^k different possible values. The table again represents the average of 100 trials per entry.

Table IV demonstrates that the expected deficiency of Least strategy is indeed approximately ℓ^k when n is sufficiently large, as shown in Theorem 8. It would be helpful to have a better understanding of both the Simple and Least strategies, which both appear to enjoy a range where performance increases as n increases. The data also suggests that while the Least strategy sometimes performs much better than the Simple strategy, with other parameter settings their performance is comparable.

VII. CONCLUSION

We have argued for the study of average-case performance of floating codes, suggesting a general model and considering restricted cases in order to obtain some specific initial results. We have found simple, asymptotically optimal codes for storing two bit values under the model that the first bit is the next to flip with probability p. We have also found that Gray codes provide a potentially useful building block in the design of such codes, combining simplicity with strong performance. Our initial work suggests that designing floating codes for expected performance is a potentially feasible approach that could improve practical performance.

We conclude with a large number of open questions.

- Are there versions of the code optimization problem we have described that are NP-hard or harder?
- Can one find efficient approaches to find optimal floating codes for expected performance? Or approaches that are at least efficient for small parameters?
- Our Gray code constructions are vulnerable to patterned sequences of bit changes. Can we modify our codes to avoid this problem with little additional cost?
- Can we find expressions for lower bounds on the cost as a function of k, ℓ, n, and the Markov chain on the value variables? Can we find specific lower bounds for the Markov chain where the ith bit is the next to flip with probability p_i?
- Under what circumstances is there wasted space, in the sense that there are cell state vectors that are never reached in the optimal solution? Are there other possible utilizations for such cell states?
- What gains are possible considering small families of codes, instead of single codes, where one of the codes from the family can be chosen each time a reset occurs?
- Can we find constructions for larger values of the parameters k, ℓ, and n?
- What results can be obtained for expected performance in similar settings using error-correction [12] or buffer coding [3]?

REFERENCES

[1] R. Ahlswede and Z. Zhang, "Coding for write-efficient memory," *Inf. Computat.*, pp. 80–97, 1989.
[2] R. Bez, E. Camerlenghi, A. Modelli, and A. Visconti, "Introduction to flash memory," *Proc. IEEE*, vol. 91, no. 4, pp. 489–502, 2003.
[3] V. Bohossian, A. Jiang, and J. Bruck, "Buffer coding for asymmetric multi-level memory," in *Proc. IEEE Int. Symp. Inf. Theory*, 2007, pp. 1186–1190.
[4] Y. Cassuto, M. Schwartz, V. Bohossian, and J. Bruck, "Codes for multi-level flash memories: Correcting asymmetric limited-magnitude errors," in *Proc. IEEE Int. Symp. Inf. Theory*, 2007, pp. 1176–1180.
[5] A. Fiat and A. Shamir, "Generalized 'write-once' memories," *IEEE Trans. Inf. Theory*, vol. IT-30, pp. 470–480, 1984.
[6] H. Finucane and M. Mitzenmacher, Worst-Case and Average-Case Floating Codes for Flash Memory Harvard Comput. Sci. Tech. Rep. TR-04-09.
[7] F. 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. Inf. Theory*, vol. 45, no. 1, pp. 308–313, 1999.
[8] F. Fu and R. W. Yeung, "On the capacity and error-correcting codes of write-efficient memories," *IEEE Trans. Inf. Theory*, vol. 46, no. 7, pp. 2299–2314, 2000.
[9] F. Gray, "Pulse Code Communications," U.S. Patent 2632058, Mar. 1953.
[10] C. Heegard, "On the capacity of permanent memory," *IEEE Trans. Inf. Theory*, vol. IT-31, pp. 34–42, 1985.
[11] C. Heegard and A. El Gamal, "On the capacity of computer memory with defects," *IEEE Trans. Inf. Theory*, vol. IT-29, pp. 731–739, 1983.
[12] A. Jiang, "On the generalization of error-correcting WOM codes," in *Proc. IEEE Int. Symp. Inf. Theory*, 2007, pp. 1391–1395.

[13] A. Jiang, V. Bohossian, and J. Bruck, "Floating codes for joint information storage in write asymmetric memories," in *Proc. IEEE Int. Symp. Inf. Theory*, 2007, pp. 1166–1170.
[14] A. Jiang and J. Bruck, "Joint coding for flash memory storage," in *Proc. IEEE Int. Symp. Inf. Theory*, 2008, pp. 1741–1745.
[15] A. Jiang, M. Landberg, M. Schwartz, and J. Bruck, "Universal rewriting in constrained memories," in *Proc. IEEE Int. Symp. Inf. Theory*, 2009, to be published.
[16] A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck, "Rank modulation for flash memories," in *Proc. IEEE Int. Symp. Inf. Theory* , 2008, pp. 1731–1735.
[17] H. Mahdavifar, P. Siegel, A. Vardy, J. Wolf, and E. Yaakobi, "A nearly optimal construction for flash codes," in *Proc. IEEE Int. Symp. Inf. Theory*, 2009, to be published.
[18] V. Malyshev and M. Menshikov, "Ergodicity, continuity and analyticity of countable Markov chains," *Trans. Moscow Math. Soc.*, vol. 39, pp. 3–48, 1979.
[19] S. Meyn and R. Tweedie*, its Markov Chains and Stochastic Stability*. New York: Springer, 1996.
[20] M. Mitzenmacher and E. Upfal*, Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge, U.K.: Cambridge Univ. Press, 2005.
[21] R. L. Rivest and A. Shamir, "How to reuse a 'write-once" memory," *Inf. Control*, vol. 55, pp. 227–231, 1984.
[22] G. Simonyi, "On write-unidirectional memory codes," *IEEE Trans. Inf. Theory*, vol. 35, no. 3, pp. 663–669, 1989.
[23] J. K. Wolf, A. D. Wyner, J. Ziv, and J. Körner, "Coding for a write-once memory," *AT& T Bell Lab. Tech. J.*, vol. 63, no. 6, pp. 1089–1112, 1984.
[24] E. Yaakobi, A. Vardy, P. Siegel, and J. Wolf, "Multidimensional flash codes," in *Proc. 2008 Allerton Conf.*, 2008.

Flavio Chierichetti received the Bachelor and Master of Science degrees in computer science from Sapienza University of Rome, Italy, in 2004 and 2006, respectively.

He is currently pursuing the Ph.D. degree in computer science with the same University. His main interests are algorithms, probability, and the dynamics of the Internet and social networks.

Hilary Finucane received the degree in mathematics (*magna cum laude*) from Harvard University, Cambridge, MA, in 2009.

Next year, she will begin graduate studies in theoretical computer science at the Weizmann Institute.

Zhenming Liu received the Bachelor of Engineering degree in computer science with a minor in mathematics from the Hong Kong University of Science and Technology in 2005, and a Master of Science degree in computer science from Harvard University, Cambridge, MA, in 2007.

He is now pursuing the Ph.D. degree from the Harvard School of Engineering and Applied Science. His research interests are theory of computation and information theory.

Michael Mitzenmacher (M'95) graduated (*summa cum laude*) with a degree in mathematics and computer science from Harvard University, Cambridge, MA, in 1991. After studying math for a year in Cambridge, England, on the Churchill Scholarship, he received the Ph.D. degree in computer science from the University of California, Berkeley, in 1996.

He then joined the Digital Systems Research Center until joining Harvard University in 1999, where he is currently a Professor of Computer Science with the School of Engineering and Applied Sciences. He has authored or coauthored more than 140 conference and journal publications on a variety of topics, including Internet algorithms, hashing, load-balancing, erasure codes, error-correcting codes, compression, bin-packing, and power laws. His textbook on probabilistic techniques in computer science titled, *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*, coauthored with Eli Upfal, was published in 2005 by Cambridge University Press.

Dr. Mitzenmacher received the 2002 IEEE Information Theory Society Best Paper Award for his work on low-density parity-check codes.