Generalized Gray Codes for Local Rank Modulation

Eyal En Gad, Michael Langberg, *Member, IEEE*, Moshe Schwartz, *Senior Member, IEEE*, and Jehoshua Bruck, *Fellow, IEEE*

***Abstract*—We consider the local rank-modulation scheme, in which a sliding window going over a sequence of real-valued variables induces a sequence of permutations. Local rank-modulation is a generalization of the rank-modulation scheme, which has been recently suggested as a way of storing information in flash memory. We study gray codes for the local rank-modulation scheme in order to simulate conventional multilevel flash cells while retaining the benefits of rank modulation. Unlike the limited scope of previous works, we consider code constructions for the entire range of parameters including the code length, sliding-window size, and overlap between adjacent windows. We show that the presented codes have asymptotically optimal rate. We also provide efficient encoding, decoding, and next-state algorithms.**

***Index Terms*—Flash memory, gray code, local rank modulation, permutations, rank modulation.**

I. Introduction

With the recent application to flash memories, the rank-modulation scheme has gained renewed interest as evident in the recent series of papers [8], [14], [15], [21], [24]. In the conventional modulation scheme used in flash-memory cells, the absolute charge level of each cell is quantized to one of q discrete levels, resulting in a single demodulated symbol from an alphabet of size q. In contrast, in the rank-modulation scheme a group of n flash cells comprise a single virtual cell storing a symbol from an alphabet of size $n!$, where each symbol is assigned a distinct configuration of relative charge levels in the n cells. Thus, there is no more need for threshold values to distinguish between various stored symbols, which mitigates the effects of retention in flash cells (slow charge leakage). In addition, if we allow only a simple programming (charge-injection) mechanism called "push-to-the-top," whereby a single cell is driven above all others in terms of charge level (see definition in Section II-B), then no overprogramming can occur, a problem which considerably slows down programming in conventional multilevel flash cells.

Rank modulation has been studied intermittently since the early works of Slepian [20] (later extended in [1]), in which permutations were used to digitize vectors from a time-discrete memoryless Gaussian source, and Chadwick and Kurz [5], in which permutations were used in the context of signal detection over channels with non-Gaussian noise (especially impulse noise). Other works on the subject include [1]–[4], [6], [7]. More recently, permutations were used for communicating over powerlines (for example, see [23]), and for modulation schemes for flash memory [14], [15], [21], [24].

One drawback to the rank-modulation scheme is the fact that we need to reconstruct the permutation induced by the relative charge levels of the participating cells. If n cells are involved, at least $\Omega(n \log n)$ comparisons are needed, which might be too high for some applications. It was therefore suggested in [8] and [24] that only *local* comparisons be made, creating a sequence of small induced permutations instead of a single all-encompassing permutation. This obviously restricts the number of distinct configurations, and thus, reduces the size of the resulting alphabet as well. In the simplest case, requiring the least amount of comparisons, the cells are located in a 1-D array and each cell is compared with its two immediate neighbors requiring a single comparator between every two adjacent cells [8].

Yet another drawback of the rank-modulation scheme is the fact that distinct n charge levels are required for a group of n physical flash cells. Therefore, restricted reading resolution prohibits the use of large values of n. However, when only local views are considered, distinct values are required only within a small local set of cells, thus enabling the use of large groups of cells with local rank modulation.

An important application for rank-modulation in the context of flash memory was described in [14]: a set of n cells, over which the rank-modulation scheme is applied, is used to simulate a single conventional multilevel flash cell with $n!$ levels corresponding to the alphabet $\{0, 1, \dots, n! - 1\}$. The simulated cell supports an operation which raises its value by 1 modulo $n!$. This is the only required operation in many rewriting schemes for flash memories (see [12], [13], [25]), and it is realized in [14] by a gray code traversing the $n!$ states where, physically, the transition between two adjacent states in the gray code is achieved by using a single "push-to-the-top" operation.

The gray code [11] was first introduced as a sequence of distinct binary vectors of fixed length, where every adjacent pair differs in a single coordinate. It has since been generalized to sequences of distinct states $s_1, s_2, \dots, s_k \in S$ such that for every $i < k$ there exists a function in a predetermined set of transitions $\tau \in T$ such that $s_{i+1} = \tau(s_i)$ (see [18] for an excellent survey). In the context of [14], the state space consisted of

Manuscript received February 17, 2013; revised June 09, 2013; accepted June 11, 2013. Date of publication June 13, 2013; date of current version September 11, 2013. This work was supported in part by the ISF Grant 134/10, in part by the ISF Grant 480/08, in part by the Open University of Israel's Research Fund under Grant 46114, in part by the NSF under Grant ECCS-0802107, in part by the NSF under Grant CIF-1218005, in part by the United States–Israel Binational Science Foundation (BSF) under Grant 2010075, and in part by a gift from Northrop Grumman.

E. En Gad and J. Bruck are with the Department of Electrical Engineering, California Institute of Technology, Pasadena, CA 91125 USA (e-mail: eengad@caltech.edu; bruck@paradise.caltech.edu).

M. Langberg is with the Department of Mathematics and Computer Science, The Open University of Israel, Raanana 43107, Israel (e-mail: mikel@openu.ac.il).

M. Schwartz is with the Department of Electrical and Computer Engineering, Ben-Gurion University, Beer Sheva 84105, Israel (e-mail: schwartz@ee.bgu.ac.il).

Communicated by I. F. Blake, Associate Editor for Complexity and Cryptography.

Digital Object Identifier 10.1109/TIT.2013.2268534

permutations over n elements, and the "push-to-the-top" operations were the allowed transitions. This operation was studied since it is a simple programming operation which is both quick and eliminates the overprogramming problem. We also note that generating permutations using "push-to-the-top" operations is of independent interest, called "nested cycling" in [19] (see also references therein), motivated by a fast "push-to-the-top" operation (cycling) available on some computer architectures.

Other generalizations of gray codes for rank modulation include snake-in-the-box codes for rank modulation [26], as well as gray codes for local rank modulation, which were studied in [8].

Having considered the two extremes: full rank modulation with a single permutation of n cells, and extreme local rank modulation with a sequence of n permutations over two elements, the question of whether any middle-road solutions exist remains open. We address this question by considering the generalized local rank-modulation scheme. In this scheme, a sequence of several permutations of a given size provide the local views into the ranking of the cells. We construct gray codes for this scheme which asymptotically achieve the maximum possible rate, and consider efficient encoding/decoding algorithms, as well as efficient next-state computation.

The rest of the paper is organized as follows. In Section II, we give preliminary definitions and notation. In Section III, we present a new construction for optimal local rank modulation for general degrees of locality. We conclude with a discussion in Section IV.

II. Definitions and Notation

We shall now proceed to introduce the notation and definitions pertaining to local rank modulation and gray codes. We will generally follow the notation introduced in [8].

A. Local Rank Modulation

Assume we have a set of t flash memory cells which we number $0, 1, \dots, t-1$. Let us consider a sequence of t real-valued variables, $\mathbf{c} = (c_0, c_1, \dots, c_{t-1}) \in \mathbb{R}^t$, where c_i denotes the charge level measured in the ith flash memory cell. We further assume $c_i \neq c_j$ for all $i \neq j$. The t variables induce a permutation $f_{\mathbf{c}} \in S_t$, where S_t denotes the set of all permutations over $[t] = \{0, 1, 2, \dots, t-1\}$. The permutation $f_{\mathbf{c}}$ is defined as

$$f_{\mathbf{c}}(i) = |\{j \mid c_j < c_i\}|\,.$$

Thus, $f_{\mathbf{c}}(i)$ is the rank of the ith cell in ascending order. This ranking is equivalent to the permutation described in [8].

We now turn to consider a larger set of $n \geqslant t$ flash memory cells. Again, we have a sequence of n variables, $\mathbf{c} = (c_0, c_1, \dots, c_{n-1}) \in \mathbb{R}^n$ where c_i denotes the measured charge level in the ith flash memory cell. We define a window of size t at position p to be

$$\mathbf{c}_{p,t} = (c_p, c_{p+1}, \dots, c_{p+t-1})$$

where the indices are taken modulo n, and also $0 \leqslant p \leqslant n-1$, and $1 \leqslant t \leqslant n$. We now define the (s,t,n)*-local rank-modulation (LRM) scheme*, which we do by defining the *demodulation* process. Let $s \leqslant t \leqslant n$ be positive integers, with $s|n$. Given a sequence of n distinct real-valued variables, $\mathbf{c} = (c_0, c_1, \dots, c_{n-1})$, the demodulation maps $\mathbf{c}$ to the sequence of n/s permutations from S_t as follows:

$$\mathbf{f}_{\mathbf{c}} = (f_{\mathbf{c}_{0,t}}, f_{\mathbf{c}_{s,t}}, f_{\mathbf{c}_{2s,t}}, \dots, f_{\mathbf{c}_{n-s,t}}). \tag{1}$$

Loosely speaking, we scan the n variables using windows of size t positioned at multiples of s and write down the permutations from S_t induced by the *local* views of the sequence.

In the context of flash-memory storage devices, we shall consider the n variables, $\mathbf{c} = (c_0, c_1, \dots, c_{n-1})$, to be the charge-level readings from n flash cells. The demodulated sequence $\mathbf{f}_{\mathbf{c}}$ will stand for the original information which was stored in the n cells. This approach will serve as the main motivation for this paper, as it was also for [8], [14], [15], [21], [24]. See Fig. 1 for an example of a demodulation of a $(3,5,12)$-locally rank-modulated signal.

We say a sequence $\mathbf{f}$ of n/s permutations over S_t is (s,t,n)*-LRM realizable* if there exists $\mathbf{c} \in \mathbb{R}^n$ such that $\mathbf{f} = \mathbf{f}_{\mathbf{c}}$, i.e., it is the demodulated sequence of $\mathbf{c}$ under the (s,t,n)-LRM scheme. Except for the degenerate case of $s = t$, not every sequence is realizable. For example, if $s < t$ and $f_{\mathbf{c}_{i\cdot s,t}}$ is the identity permutation (i.e., $f_{\mathbf{c}_{i\cdot s,t}}(i) = i$) then for all $1 \leqslant j < n$ we have $c_j < c_{j+1}$, but also $c_n < c_1$, which is impossible.

We denote the set of all (s,t,n)-LRM realizable permutation sequences as $\mathcal{R}(s,t,n)$. In a later part of this section, we show that the number of states representable by an (s,t,n)-LRM scheme, i.e., the size of $\mathcal{R}(s,t,n)$, is roughly $(t\cdot(t-1)\cdot\dots\cdot(t-s+1))^{n/s}$ (this fact is also stated in [24]).

While any $\mathbf{f} \in \mathcal{R}(s,t,n)$ may be represented as a sequence of n/s permutations over S_t, a more succinct representation is possible based on the (mixed-radix) factoradic notation system (see [16] for the earliest-known definition, and [14] for a related use): we can represent any permutation $f = [f(0), \dots, f(t-1)] \in S_t$ with a sequence of digits $d_{t-1}, d_{t-2}, \dots, d_1, d_0$ (note the reversed order of indices), where $d_{t-1-i} \in \mathbb{Z}_{t-i}$, and d_{t-1-i} counts the number of entries $f(j)$ for $j > i$ which are of value lower than $f(i)$, i.e.,

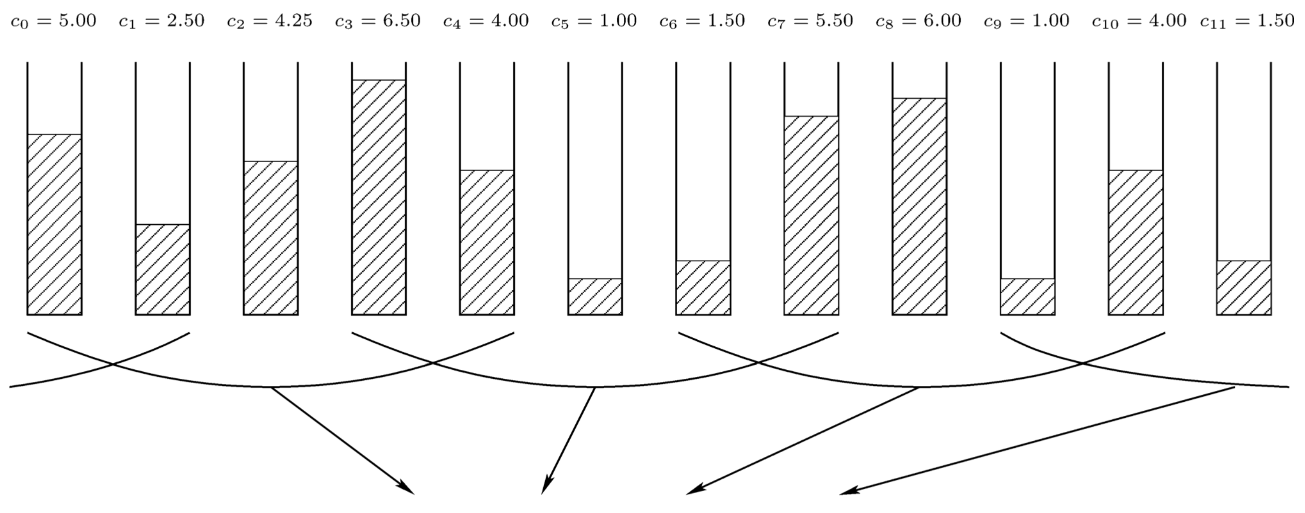
$$d_{t-1-i} = |\{j > i \mid c_j < c_i\}|.$$

We call d_{t-1} the *most-significant digit* and d_0 the *least-significant digit*. If $f = f_{\mathbf{c}}$ for some $\mathbf{c} \in \mathbb{R}^t$, then the factoradic representation is easily seen to be equivalent to counting the number of cells to the right of the ith cell which are with lower charge levels.

Continuing with the succinct representation, we now contend that due to the overlap between local views, we can represent each of the local permutations $f_{\mathbf{c}_{i\cdot s,t}}$ using only the s most-significant digits in their factoradic notation. We denote this (partial) representation as $\bar{f}_{\mathbf{c}_{i\cdot s,t}}$ and call it the *condensed factoradic* representation. Accordingly, we define,

$$\bar{\mathbf{f}}_{\mathbf{c}} = (\bar{f}_{\mathbf{c}_{0,t}}, \bar{f}_{\mathbf{c}_{s,t}}, \bar{f}_{\mathbf{c}_{2s,t}}, \dots, \bar{f}_{\mathbf{c}_{n-s,t}}),$$

and the set of all such presentations as $\bar{\mathcal{R}}(s,t,n)$. Thus, for example, the configuration of Fig. 1 would be represented by $((3,0,1),(4,2,0),(1,2,2),(0,2,0))$. Since throughout the rest


$c_0 = 5.00$ $c_1 = 2.50$ $c_2 = 4.25$ $c_3 = 6.50$ $c_4 = 4.00$ $c_5 = 1.00$ $c_6 = 1.50$ $c_7 = 5.50$ $c_8 = 6.00$ $c_9 = 1.00$ $c_{10} = 4.00$ $c_{11} = 1.50$

Demodulation: $\mathbf{f_c} = ([3,0,2,4,1],[4,2,0,1,3],[1,3,4,0,2],[0,3,1,4,2])$
Condensed factoradic: $\bar{\mathbf{f}}_\mathbf{c} = ((3,0,1),(4,2,0),(1,2,2),(0,2,0))$

Fig. 1. Demodulating a $(3,5,12)$-locally rank-modulated signal.

of the paper, we shall deal with the condensed factoradic representation only, we omit from now on the term "condensed."

Lemma 1: For all $1 \leqslant s \leqslant t \leqslant n$,

$$\left|\bar{\mathcal{R}}(s,t,n)\right| \leqslant |\mathcal{R}(s,t,n)| \leqslant (t-s)! \cdot \left(\frac{t!}{(t-s)!}\right)^{\frac{n}{s}}.$$

Proof: As $\bar{\mathcal{R}}(s,t,n) = \{\bar{\mathbf{f}}_\mathbf{c} \mid \mathbf{f_c} \in \mathcal{R}(s,t,n)\}$, we have that $\left|\bar{\mathcal{R}}(s,t,n)\right| \leqslant |\mathcal{R}(s,t,n)|$. For the upper bound, assume we fix the permutation induced by the first $t-s$ cells, where there are $(t-s)!$ ways of doing so. It follows that there are at most $t!/(t-s)!$ ways of choosing $f_{\mathbf{c}_{n-s,t}}$, and then the same bound on the number of ways of choosing $f_{\mathbf{c}_{n-2s,t}}$, and continuing all the way up to $f_{\mathbf{c}_{0,t}}$ we obtain the desired bound. ■

When $s = t = n$, the (n,n,n)-LRM scheme degenerates into a single permutation from S_n. This was the case in most of the previous works using permutations for modulation purposes. A slightly more general case, $s = t < n$ was discussed by Ferreira *et al.* [9] in the context of permutation trellis codes, where a binary codeword was translated tuple-wise into a sequence of permutation with no overlap between the tuples. An even more general case was defined by Wang *et al.* [24] (though in a slightly different manner where indices are not taken modulo n, i.e., with no wrap-around). In [24], the sequence of permutations was studied under a charge-difference constraint called *bounded rank-modulation*, and mostly with parameters $s = t-1$, i.e., an overlap of one position between adjacent windows. Finally, using the same terminology as this paper, the case of $(1,2,n)$-LRM was considered in [8].

B. Gray Codes

A *gray code*, G, is a sequence of distinct states (codewords), $G = g_0, g_1, \dots, g_{N-1}$, from an ambient state space, $g_i \in S$, such that adjacent states in the sequence differ by a "small" change. What constitutes a "small" change usually depends on the code's application.

Since we are interested in building gray codes for flash memory devices with the (s,t,n)-LRM scheme, the ambient space is $\mathcal{R}(s,t,n)$, which is the set of all realizable sequences under (s,t,n)-LRM.

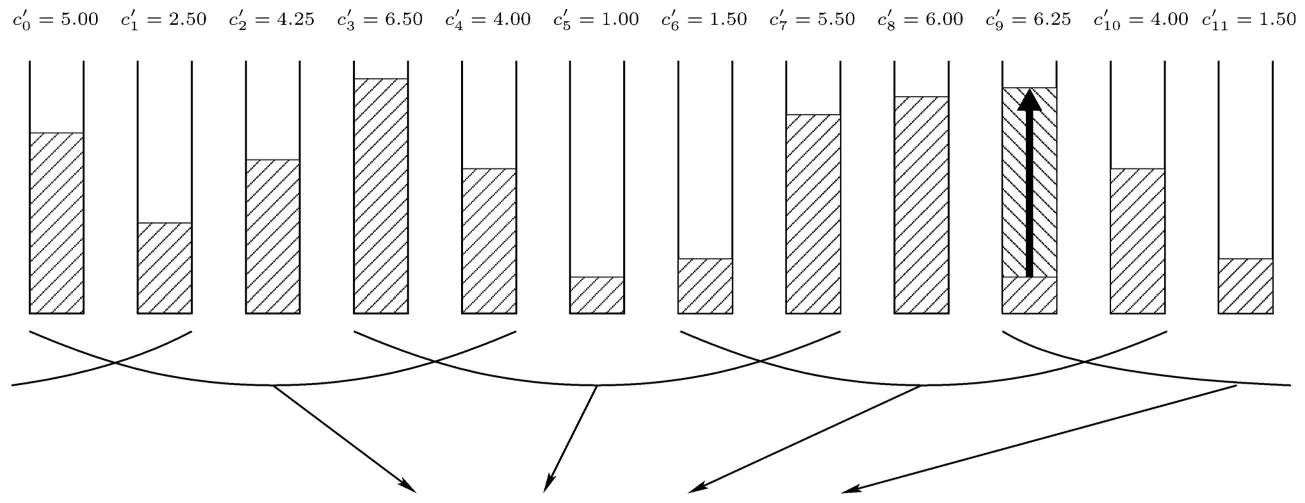
The transition between adjacent states in the gray code is directly motivated by the flash memory application, and was first described and used in [14], and later also used in [8]. This transition is the "push-to-the-top" operation, which takes a single flash cell and raises its charge level above all others.

In our case, however, since we are considering a *local* rank-modulation scheme, the "push-to-the-top" operation merely raises the charge level of the selected cell above those cells which are comparable with it.

In Fig. 2, we see the example signal of Fig. 1 after a "push-to-the-top" operation performed on the ninth cell. The cells participating with the ninth cell in local permutations are $6,7,8,10,11,0,1$, i.e., from cell 6 to cell 1 with wrap-around. Thus, the charge level of the ninth cell was pushed above that of cells 6 through 1 (with wrap-around). We do note that the new charge level of the ninth cell is not above that of all other cells since the third cell still has a higher level. However, the third cell is incomparable (i.e., does not participate in a local permutation) with the ninth cell. Fig. 2 also shows the demodulation and condensed factoradic representation of the new configuration. By comparing with Fig. 1, we note that a single "push-to-the-top" operation can change several digits in the demodulated sequence and in the factoradic notation.

We now provide a precise definition of the "push-to-the-top" operation in the local rank-modulation scheme. Assume we have n flash memory cells with charge levels $\mathbf{c} = (c_0, \dots, c_{n-1}) \in \mathbb{R}^n$. We say cell j is comparable with cell j' if they both participate in some window together. We shall denote these cells as the cells numbered $l(j), l(j)+1, \dots, r(j)$ where one can easily verify that

$$l(j) = s\left\lceil \frac{j-t+1}{s} \right\rceil \bmod n,$$
$$r(j) = \left(s\left\lfloor \frac{j}{s} \right\rfloor + (t-1)\right) \bmod n.$$



Demodulation: $\mathbf{f}_{\mathbf{c}'} = ([3,0,2,4,1],[4,2,0,1,3],[0,2,3,4,1],[4,2,0,3,1])$
Condensed factoradic: $\bar{\mathbf{f}}_{\mathbf{c}'} = ((3,0,1),(4,2,0),(0,1,1),(4,2,0))$

Fig. 2. "Push-to-the-top" operation performed on the ninth cell.

We note that the indices $l(j), l(j)+1, \ldots, r(j)$ should be taken modulo n. Continuing the example of Fig. 2, for $s=3$ and $t=5$ we have $l(8) = 3\lceil(8-5+1)/3\rceil = 6$, and $r(10) = (3\lfloor 10/3\rfloor + 5 - 1) \equiv 1 \pmod{12}$, i.e., the left-most cell comparable with cell 8 is cell 6, while the right-most cell comparable with cell 10 is cell 1.

A "push-to-the-top" operation performed on cell j changes the cell levels to $\mathbf{c}' = (c'_0, \ldots, c'_{n-1}) \in \mathbb{R}^n$ defined by

$$c'_i = \begin{cases} c_i & i \neq j, \\ \max\left\{c_{l(i)}, \ldots, c_{r(i)}\right\} + \epsilon & i = j, \end{cases}$$

where $\epsilon > 0$ denotes a small charge difference which is a parameter of the physical charge-placement mechanism. Namely, the charge level of cell j is pushed above the charge levels of cells comparable with it.

We can now move from the physical level to the logical level of the demodulated signal. With the above notation, $\mathbf{c}'$ was achieved from $\mathbf{c}$ by a "push-to-the-top" operation on cell j. Let $\mathbf{f}_{\mathbf{c}}$ and $\mathbf{f}_{\mathbf{c}'}$ stand for the demodulated sequence of permutations from $\mathbf{c}$ and $\mathbf{c}'$, respectively. We then say $\mathbf{f}_{\mathbf{c}'}$ was achieved from $\mathbf{f}_{\mathbf{c}}$ by a single "push-to-the-top" operation on cell j. Thus, we define the set of allowed transitions as $T = \{\tau_0, \tau_1, \ldots, \tau_{n-1}\}$, which is a set of functions, $\tau_j : \mathcal{R}(s,t,n) \to \mathcal{R}(s,t,n)$, where τ_j represents a "push-to-the-top" operation performed on the jth cell.

Definition 1: A *Gray code* G for (s,t,n)-LRM (denoted (s,t,n)-LRMGC) is a sequence of distinct codewords, $G = g_0, g_1, \ldots, g_{N-1}$, where $g_i \in \mathcal{R}(s,t,n)$. For all $0 \leqslant i \leqslant N-2$, we further require that $g_{i+1} = \tau_j(g_i)$ for some j. If $g_0 = \tau_j(g_{N-1})$ for some j, then we say the code is *cyclic*. We call N the *size* of the code, and say G is *optimal* if $N = |\mathcal{R}(s,t,n)|$.

Definition 2: A family of codes, $\{G_i\}_{i=1}^{\infty}$, where G_i is an (s,t,n_i)-LRMGC of size N_i, $n_{i+1} > n_i$, is *asymptotically rate-optimal* if

$$\lim_{i\to\infty} \frac{\log_2 N_i}{\log_2 |\mathcal{R}(s,t,n_i)|} = 1.$$

III. Gray Codes for (s,t,n)-LRM

In this section, we present efficiently encodable and decodable asymptotically rate-optimal gray codes for (s,t,n)-LRM. A rough description of the proposed construction follows. First we partition the n cells into m blocks, each containing n/m cells. To simplify the presentation we set $n = m^2$, implying that we have m blocks, each of size m. Denote the cells in block i by $\mathbf{c}_i$. For each block $\mathbf{c}_i$, we will use the factoradic representation $\bar{\mathbf{f}}_{\mathbf{c}_i}$ to represent permutations, without wrap-around at the block level (the wrap-around is only for the entire codeword). Namely, each and every block can be thought of as an element of an alphabet $\Sigma = \{v_0, \ldots, v_{V-1}\}$ of a size denoted by V.

Now, consider any De-Bruijn sequence S of order $m-1$ over Σ (of period V^{m-1}). Namely, S will consist of a sequence of V^{m-1} elements $v_{s_0}, v_{s_1}, \ldots, v_{s_{V^{m-1}-1}}$ over Σ such that the subsequences $v_{s_i}, \ldots, v_{s_{i+m-2}}$ of S *cover* all $(m-1)$-tuples of Σ exactly once, subindices of s are taken modulo V^{m-1}. We shall conveniently choose $\Sigma = \mathbb{Z}_V$. Such sequences S exist, e.g., [10].

The construction of the gray code G will have two phases. First we construct the so-called *anchor* elements in G, denoted as $\bar{G} = \{g_0, \ldots, g_{L-1}\}$. The elements of $\bar{G}$ will consist of a cyclic gray code over Σ^m. That is, the difference between each g_i and g_{i+1} in $\bar{G}$ will be in only one out of the m characters (from Σ) in g_i. Specifically, the code $\bar{G}$ will be derived from the De-Bruijn sequence S as follows: we set g_0 to be the first m elements of S, and in the transition from g_i to g_{i+1} we change v_{s_i} to $v_{s_{i+m}}$. The code $\bar{G}$ is detailed below:

$$\begin{array}{rcccccc}
g_0 & = & v_{s_{m-1}} & v_{s_{m-2}} & \cdots & v_{s_1} & \underline{v_{s_0}} \\
g_1 & = & v_{s_{m-1}} & v_{s_{m-2}} & \cdots & \underline{v_{s_1}} & v_{s_m} \\
g_2 & = & v_{s_{m-1}} & v_{s_{m-2}} & \cdots & v_{s_{m+1}} & v_{s_m} \\
 & & & & \vdots & & \\
g_{L-2} & = & v_{s_{L-1}} & \underline{v_{s_{L-2}}} & \cdots & v_{s_1} & v_{s_0} \\
g_{L-1} & = & \underline{v_{s_{L-1}}} & v_{s_{m-2}} & \cdots & v_{s_1} & v_{s_0}
\end{array}$$

where $L = \operatorname{lcm}(m, V^{m-1})$, the subindices of s are taken modulo V^{m-1}, and the underline is an imaginary marking distinguishing the block which is about to change.

With the imaginary marking of the underline, the code $\bar{G}$ is clearly a gray code over Σ^m due to the properties of the De-Bruijn sequence S. However, $\bar{G}$ is not a gray code over $\mathcal{R}(s,t,n)$, as the transitions between the anchors g_i and g_{i+1} require changing the entries of an entire block, which may involve many push-to-the-top operations. We thus refine $\bar{G}$ by adding additional elements between each pair of adjacent anchors from $\bar{G}$ that allow us to move from the block configuration in g_i to that in g_{i+1} by a series of push-to-the-top operations.

Each block is eventually represented in factoradic notation by a sequence of digits. For the construction to work, it is crucial to be able to identify, for each element in $\bar{G}$, which block is in the process of being changed. To this end, and for other technical reasons, we will add some auxiliary digits to each block (by adding auxiliary flash memory cells). These digits are referred to as *noninformation* digits. Loosely speaking, in each block of size m the last $t+1$ digits (approximately[1]) will be noninformation digits. The last two digits will be used to mark which block is being currently changed, while the $t-1$ digits before them will act as a *buffer zone* that allows the successful transformation between anchors. In our setting, the $t+1$ noninformation digits will be negligible with respect to the remaining $m-(t+1)$ information digits, allowing the code $\bar{G}$ to have asymptotically optimal rate.

Construction 1: We consider the (s,t,n)-LRM scheme, where

$$n = m^2, \qquad m \geqslant t+2, \qquad s \mid m.$$

Let $\{v_0, v_1, \ldots, v_{V-1}\}$ be a set of V distinct mixed-radix vectors of length m. Each vector v_i is representing m/s local permutations, where each permutation is represented by the s most-significant digits of its factoradic notation. Therefore,

$$v_i \in \left(\mathbb{Z}_t \times \mathbb{Z}_{t-1} \times \cdots \times \mathbb{Z}_{t-(s-1)}\right)^{m/s}.$$

Let us denote

$$\delta = \left\lceil \frac{t+2}{s} \right\rceil - 1.$$

The values of the last $s\delta$ digits (that represent the last δ local permutations) of each v_i do not play a role in the representation of the stored data and are called *noninformation digits*. By abuse of notation, two vectors agreeing on the first $m - s\delta$ digits will be said to represent the same value. Furthermore, when a block is said to represent some value v_i, we mean that its first $m - s\delta$ digits agree with those of v_i. Therefore, we set

$$V = \left(\frac{t!}{(t-s)!}\right)^{\frac{m}{s}-\delta}.$$

We also denote $L = \operatorname{lcm}(m, V^{m-1})$.

Let S be a De-Bruijn sequence of order $m-1$ over the alphabet $\mathbb{Z}_V$,

$$S = s_0, s_1, \ldots, s_{V^{m-1}-1},$$

i.e., S is of length V^{m-1} and $s_i \in \mathbb{Z}_V$. The gray code $\bar{G}$ of anchor vectors is a sequence

$$\bar{G} = g_0, g_1, \ldots, g_{L-1}$$

of L mixed-radix vectors of length $n = m^2$. Each vector is formed by a concatenation of m blocks of length m. For $k \in [m]$, we say that block k corresponds to the cells with indices $km, km+1, \ldots, (k+1)m-1$. We set g_0 to be the concatenation of the first m elements of S, such that for each $k \in [m]$, block k represent the vector $v_{s_{m-1-k}}$:

$$g_0 = v_{s_{m-1}} v_{s_{m-2}} \cdots v_{s_1} v_{s_0}.$$

Between the anchors g_i and g_{i+1}, the block that represents the vector v_{s_i} is transformed into the vector $v_{s_{i+m}}$. The resulting gray code $\bar{G}$ of anchor vectors is therefore

$$\begin{array}{rcccccc}
g_0 &=& v_{s_{m-1}} & v_{s_{m-2}} & \cdots & v_{s_1} & \underline{v_{s_0}} \\
g_1 &=& v_{s_{m-1}} & v_{s_{m-2}} & \cdots & \underline{v_{s_1}} & v_{s_m} \\
g_2 &=& v_{s_{m-1}} & v_{s_{m-2}} & \cdots & v_{s_{m+1}} & v_{s_m} \\
 & & & & \vdots & & \\
g_{L-2} &=& v_{s_{L-1}} & \underline{v_{s_{L-2}}} & \cdots & v_{s_1} & v_{s_0} \\
g_{L-1} &=& \underline{v_{s_{L-1}}} & v_{s_{m-2}} & \cdots & v_{s_1} & v_{s_0}
\end{array}$$

where the underline denotes the block that is about to change in the transition to the following anchor vector.

Within each of the m blocks comprising any single anchor, the $(m-2)$nd digit (the next-to-last digit—a noninformation digit) corresponds to a cell that is pushed-to-the-top in all blocks except for the "underlined" block (i.e., the block which is about to change). For brevity, we call this digit the *underlined digit*. In the underlined block, the $(m-1)$th digit is pushed-to-the-top. All remaining noninformation digits are initialized to be of value 0.

Between any two anchors, g_i and g_{i+1}, a sequence of vectors called *auxiliary vectors* and denoted $g_i^0, g_i^1, \ldots, g_i^{\ell_i}$, is formed by a sequence of push-to-the-top operations on the cells of the changing block. The auxiliary vectors are determined by Algorithm 1 described shortly. Thus, the entire gray code G constructed is given by the sequence

$$\begin{array}{lllll}
g_0, & g_0^0, & g_0^1, & \cdots & g_0^{\ell_0}, \\
g_1, & g_1^0, & g_1^1, & \cdots & g_1^{\ell_1}, \\
\vdots & & & & \\
g_{L-1}, & g_{L-1}^0, & g_{L-1}^1, & \cdots & g_{L-1}^{\ell_{L-1}}.
\end{array}$$

In what follows we present Algorithm 1 that specifies the sequence $g_i^0, g_i^1, \ldots, g_i^{\ell_i}$ that allows us to move from anchor state g_i to state g_{i+1}. As g_i and g_{i+1} differ only in a single block (and this block is changed from representing the value v_{s_i} to $v_{s_{i+m}}$), it holds that g_i^j and $g_i^{j'}$ will differ only in the block in which g_i and g_{i+1} differ. Thus, it suffices to define in Algorithm 1 how to change a block of length m with cell values that represent v_{s_i} into a block that represents $v_{s_{i+m}}$ using push-to-the-top operations. However, we call the attention of the reader to the fact that while the change in represented value affects only one block (denoted as block k), for administrative reasons, in block

[1]For the exact value see Construction 1.

$k-1$ (modulo m), we also push a *noninformation* cell. The inputs of Algorithm 1 are the vector $v_{s_{i+m}}$ and the corresponding cell configuration $(c_{km}, c_{km+1}, \ldots, c_{(k+1)m-1})$, and its output is the *order* in which those cells are pushed in order to transform the vector represented in block k into $v_{s_{i+m}}$. Algorithm 1 also ensures that the underlined digits of blocks k and $k-1$ (modulo m) of an auxiliary vector are both not of maximal value (among the cells with which they share a window). This allows to identify whether the vector is an anchor vector or an auxiliary one. Assuming that

$$(r_0, r_1, \ldots, r_{m-1}) \in \left(\mathbb{Z}_t \times \mathbb{Z}_{t-1} \times \cdots \times \mathbb{Z}_{t-(s-1)}\right)^{m/s}$$

represents the value v_ℓ, then we say that the jth digit of v_ℓ is

$$v_\ell(j) = \begin{cases} r_j & 0 \leqslant j < m - s\delta \\ 0 & \text{otherwise,} \end{cases}$$

i.e., we always force a 0 for the noninformation digits. Finally, we restrict $l(\cdot)$ and $r(\cdot)$ by defining

$$l'(j) = \begin{cases} l(j) & 0 \leqslant l(j) \leqslant m-3 \\ 0 & \text{otherwise} \end{cases}$$
$$r'(j) = \begin{cases} r(j) & 0 \leqslant r(j) \leqslant m-3 \\ m-3 & \text{otherwise.} \end{cases}$$

Algorithm 1 Transform Block k of Configuration $\mathbf{f_c}$ From v_{s_i} to v_{s_i+m}

```
Input: current cell configuration $\mathbf{f_c}$, block number $k$, new block value $v_{s_{i+m}}$
Output: new cell configuration $\mathbf{f_{c'}}$
Push cell $m-1$ (the last cell) in block $k-1$ (modulo $m$).
$a_j \Leftarrow 0$ for all $j = 0, 1, \ldots, m-3$
$j \Leftarrow 0$
repeat
    if $v_{s_{i+m}}(j) = \sum_{i=j+1}^{r'(j)} a_i$ and $a_j = 0$ then
        if $c_{km+j} \leqslant \max_{l'(j) \leqslant j' \leqslant r'(j), a_{j'}=1} c_{km+j'}$ then
            Push the $j$th cell in block $k$ (the cell in position $km+j$).
        end if
        $a_j \Leftarrow 1$
        $j \Leftarrow l'(j)$
    else
        $j \Leftarrow j+1$
    end if
until $j = m-2$
Push cell $m-2$ (the next-to-last cell) in block $k$.
Output the resulting cell configuration $\mathbf{f_{c'}}$.
```

Algorithm 1 is strongly based on the factoradic representation of $v_{s_{i+m}}$. Let $v_{s_{i+m}}(j)$ be the jth entry in this representation. Namely, if $\mathbf{c} = (c_0, \ldots, c_{m-1})$ is a cell configuration that corresponds to $v_{s_{i+m}}$, then for each index $j \in [m]$ we let $v_{s_{i+m}}(j)$ denote the number of entries in the window corresponding to j that are in a larger position than j and are of value lower than c_j. Roughly speaking, to obtain such a configuration $\mathbf{c}$, Algorithm 1, for $j \in [m]$, *marks* each cell c_j in $\mathbf{c}$ after exactly $v_{s_{i+m}}(j)$ cells in positions larger than j (and participating in the window corresponding to j) have been marked. Here, in order to keep track of which cells should not be pushed anymore, we save an array of bits a_j for each cell in the block (initialized to 0), indicating whether the cell c_j should not be pushed anymore. If $a_j = 1$, we say that cell j is marked. Furthermore, when the cell is marked, it is also pushed-to-the-top if its value is lower than that of a cell that shares a window with it and is already marked (the value comparison can be inferred from $\mathbf{f_c}$). Since each time a cell is changed it is pushed-to-the-top, this will ensure that the resulting cell configuration $\mathbf{c}$ will have a factoradic representation corresponding to $v_{s_{i+m}}$.

We note that in order to be able to decode a state, we need to have some way to know which block is being currently changed, i.e., the imaginary underline in the anchor. We use the last two cells of each block for that purpose as described in the example below.

Example 1: Let us consider the case of $(1, 2, 16)$-LRM, i.e., $s = 1$, $t = 2$, $m = 4$, and $n = m^2 = 16$. Thus,

$$\delta = \left\lceil \frac{t+2}{s} \right\rceil - 1 = \left\lceil \frac{2+2}{1} \right\rceil - 1 = 3,$$

and so in each block, the last $s\delta = 3$ digits are noninformation digits, leaving only the first digit in each block to be an information digit.

According to the construction, we set

$$V = \left(\frac{t!}{(t-s)!}\right)^{\frac{m}{s}-\delta} = \left(\frac{2!}{(2-1)!}\right)^{\frac{4}{1}-3} = 2.$$

We therefore take a De-Bruijn sequence of order 3 and alphabet of size 2,

$$S = 0, 0, 0, 1, 0, 1, 1, 1.$$

The list of anchors is

$$\begin{aligned}
g_0 &= \mathbf{1}010\quad \mathbf{0}010\quad \mathbf{0}010\quad \underline{\mathbf{0}000}\\
g_1 &= \mathbf{1}010\quad \mathbf{0}010\quad \underline{\mathbf{0}000}\quad \mathbf{0}010\\
g_2 &= \mathbf{1}010\quad \underline{\mathbf{0}000}\quad \mathbf{1}010\quad \mathbf{0}010\\
g_3 &= \underline{\mathbf{1}000}\quad \mathbf{1}010\quad \mathbf{1}010\quad \mathbf{0}010\\
g_4 &= \mathbf{1}010\quad \mathbf{1}010\quad \mathbf{1}010\quad \underline{\mathbf{0}000}\\
g_5 &= \mathbf{1}010\quad \mathbf{1}010\quad \underline{\mathbf{1}000}\quad \mathbf{0}010\\
g_6 &= \mathbf{1}010\quad \underline{\mathbf{1}000}\quad \mathbf{0}010\quad \mathbf{0}010\\
g_7 &= \underline{\mathbf{1}000}\quad \mathbf{0}010\quad \mathbf{0}010\quad \mathbf{0}010.
\end{aligned}$$

The bold bit (the leftmost bit in each group of four) denotes the information bit, while the rest are noninformation bits. The underlined vectors are easily recognizable by the next-to-right-most (next-to-last) bit being 0.

Notice that in this example the information bit is dominated in size by the remaining bits of each block. This is an artifact of our example in which we take n to be small. For large values of n the overhead in each block is negligible with respect to the information bits.

As an example, the transition between g_1 and g_2 is (the changed positions are underlined):

$$\begin{aligned} g_1 &= 1010 \quad 0010 \quad 0000 \quad 0010 \\ g_1^0 &= 1010 \quad 00\underline{01} \quad 0000 \quad 0010 \\ g_1^1 &= 1010 \quad 000\underline{0} \quad \underline{1}000 \quad 0010 \\ g_2 &= 1010 \quad 0000 \quad 10\underline{1}0 \quad 0010. \end{aligned}$$

In this example, three cells are pushed. First, the last cell in block number 1 is pushed, according to the first line of Algorithm 1. The push affects the value of the last two digits of that block, and in such, signifies that the new vector is not an anchor. Next, the first cell of block 2 is pushed, affecting the value of both the last digit of block 1 and the first digit of block 2. Note that the last digit of block 1 is not an information bit, and it has no meaning in the decoding of the gray code. Finally, the next-to-last bit of block 2 is pushed, signifying that the new vector is an anchor.

We now address the analysis of Algorithm 1.

Lemma 2: Assuming the position of the underlined digit is known, all anchors used in Construction 1 are distinct.

Proof: The proof follows directly from the properties of the De-Bruijn sequence S and the fact that we are taking L to be the $\mathrm{lcm}(m, V^{m-1})$. ■

Lemma 3: Algorithm 1 maintains the correctness of the underlined digit in anchors (that is, the digit signifies correctly which block is about to change). In addition, between any two adjacent anchors, Algorithm 1 guarantees that the underlined digits of the changing block (block k) and its predecessor (block $(k-1) \bmod m$) are both not of maximal value (among the cells with which they share a window).

Proof: The proof is by induction. The base case follows from the construction of the first anchor element g_0. Assume g_i satisfies the inductive claim. When applying Algorithm 1 to move from anchor g_i to g_{i+1}, we start by pushing the last cell of block $k-1$. This implies that the value of the underlined cell in both block k and block $k-1$ (modulo m) are now not maximal. This state of affairs remains until the end of Algorithm 1, in which we push the next-to-the last cell in the changed block (block k). At that point in time, the underlined cell in the changed block obtains its maximal value, while block $k-1$ (that is to be changed in the next application of Algorithm 1) is of nonmaximal value. All the other underlined cells remain unchanged throughout the execution of Algorithm 1. ■

Lemma 4: All words in the code G are distinct.

Proof: First, remember that the words in G are permutation sequences, while the construction is using the factoradic notation. However, different factoradic vectors always correspond to different permutation sequences, and thus it is enough to show that the factoradic vectors are distinct. Next, by Lemmas 2 and 3, we know that all of the anchors in Construction 1 are distinct. It is left to show that adding the auxiliary vectors resulting from Algorithm 1, G remains a gray code. It is easily seen that a nonanchor codeword can never be mistaken for an anchor, and that due to the De-Bruijn sequence, two auxiliary vectors g_i^j and $g_{i'}^{j'}$ can never be the same when $i \neq i'$.

Thus, it suffices to focus only on the block being changed. In order to show that every word generated by Algorithm 1 in a single execution is distinct, we will show that every cell configuration we encounter will never be visited again. Specifically, given a configuration, we let j be the next cell that will be pushed. Since cell j is going to be pushed, there exists a cell j' such that $c_j < c_{j'}$ and $a_{j'} = 1$. After the push, $c_j > c_{j'}$. But since $a_{j'} = 1$, cell j' will not be pushed anymore, and thus in all future configurations cell j will be higher than cell j'. Therefore, all future configurations will differ from the initial one. ■

Lemma 5: Algorithm 1 terminates, and when it does, all of the cells are marked exactly once.

Proof: The index j is incremented by 1 in the algorithm's loop, unless a cell is marked. Since a cell cannot be marked more than once, the algorithm must terminate.

For each noninformation digit index $j' \in \{m - s\delta, \dots, m-3\}$, we forced $v_{s_{i+m}}(j') = 0$, and therefore each of those cells is marked the first time that $j = j'$. Now we assume by induction that for each $j' \leqslant m - s\delta$, all of the cells with indices l, $j' \leqslant l \leqslant m-3$ are marked before the algorithm terminates.

The base case, $j' = m - s\delta$, was already proved above. For the induction step, by the induction assumption, we know that all the cells in $\{j', \dots, m-3\}$ are eventually marked, and in particular, the cells in $\{j', \dots, r'(j'-1)\}$ are eventually marked. At the point where exactly $v_{s_{i+m}}(j'-1)$ of them are marked, the index j in the algorithm is guaranteed to be lowered below $j'-1$, and so, cell $j'-1$ will be marked the next time it is visited. Since the algorithm never marks a cell more than once, the claim is proved. ■

Theorem 6: Algorithm 1 changes a block representing v_{s_i} into a block representing $v_{s_{i+m}}$.

Proof: When cell j is being marked, exactly $v_{s_{i+m}}(j)$ cells from $\{j+1, \dots, r'(j)\}$ are marked with $a_j = 1$, and thus will not be pushed anymore. The rest will be pushed after and above it, and therefore its rank is exactly $v_{s_{i+m}}(j)$, as desired. ■

Lemma 7: The time complexity of Algorithm 1 is $O(tm)$.

Proof: Each cell is visited by the algorithm at most t times, once during the first visit of the algorithm, and once following each of the $t-1$ cells immediately to its right being pushed. Since each cell is pushed at most once, a full execution of the algorithm takes $O(tm)$ steps. ■

Combining all of the observations up to now, we are able to summarize with the following theorem for G from Construction 1.

Theorem 8: The code G from Construction 1 is a gray code of size at least L.

Corollary 9: For all constants $1 \leqslant s < t$, there exists an asymptotically rate-optimal family of codes, $\{G_i\}_{i=t+2}^{\infty}$, where G_i is an (s,t,n_i)-LRMGC of size N_i, $n_{i+1} > n_i$, with

$$\lim_{i\to\infty} \frac{\log_2 N_i}{\log_2 |\mathcal{R}(s,t,n_i)|} = 1.$$

Proof: We set $n_i = s^2 i^2$ for all $i \geqslant t+2$, and define L_i and V_i according to L and V in Construction 1 (where i is the index of the code in the family of codes). Then, $N_i \geqslant L_i \geqslant V_i^{si-1}$. It follows that

$$\begin{aligned}
&\lim_{i\to\infty} \frac{\log_2 N_i}{\log_2 |\mathcal{R}(s,t,n_i)|} \geqslant \\
&\geqslant \lim_{i\to\infty} \frac{(si-1)\log_2 V_i}{\log_2 \left((t-s)! \cdot \left(\frac{t!}{(t-s)!} \right)^{si^2} \right)} \\
&= \lim_{i\to\infty} \frac{(si-1)\log_2 \left(\frac{t!}{(t-s)!} \right)^{i - \lceil \frac{t+2}{s} \rceil + 1}}{\log_2 \left((t-s)! \cdot \left(\frac{t!}{(t-s)!} \right)^{si^2} \right)} \\
&= 1.
\end{aligned}$$

■

In order to use the gray codes of Construction 1, we need to have a way to associate the codewords to their indices. An encoding method should allow to identify with any index i the corresponding codeword g_i, and a decoding method should offer the reverse functionality. In addition, it is useful to also have a next-step method, that calculates the next word in the code.

A next-step method for Construction 1 is straightforward to define. Given a word g_i, consider first the case that it is an anchor vector. The next-step algorithm first identifies the block that is about to be changed, according to its nonmaximal underlined digit. Similarly, if g_i is an auxiliary vector, the next-state algorithm identifies the block that is currently being changed. With that information at hand, the algorithm continues and finds the De-Bruijn subsequence represented by g_i. According to the subsequence, the algorithm now finds the next symbol in the De-Bruijn sequence. Efficient next-state algorithms for De-Bruijn sequences, as well as encoding and decoding methods, are described in [17] and [22]. With the knowledge of the next De-Bruijn symbol, the next-state algorithm runs Algorithm 1 to find the cell that should be pushed next, and accordingly, the next state in the gray code.

It is natural to try applying the same idea for encoding and decoding as well. However, there is an obstacle that makes it less straightforward in this case. Consider the decoding function for example. By the same means as before, we can identify the index in the De-Bruijn sequence quickly. But in order to use it for identifying the index in the gray code, we would need to know the *distance* between each pair of adjacent anchors. The problem is that this distance, that is determined by the number of pushed cells in Algorithm 1, is not constant. Therefore, we would need to calculate it for each of the previous symbols in the De-Bruijn sequence. Since the length of the De-Bruijn sequence is exponential in m, this method would be inefficient. We note that this problem also applies to the construction that was presented in [8].

To tackle this obstacle, we suggest a slight modification to Construction 1, that allows for efficient encoding and decoding in the manner described above. The main purpose of the modification is to make the distance between the anchors constant. When this distance is constant, the decoder can simply multiply it with the De-Bruijn index, and find the index of the nearest anchor smaller than g_i. From there, applying Algorithm 1 completes the decoding efficiently. A similar observation holds for the encoder.

To make the distance between the anchors constant, we take the approach of using an additional counter. We aim to make the distance to always be $m+2$, i.e., exactly $m+1$ auxiliary vectors between adjacent anchors. Since Algorithm 1 creates between 1 to $m-1$ auxiliary vectors between anchors (not including the anchors), we use the counter to create between 2 to m additional auxiliary vectors. Therefore, the counter should be able to count from 0 up to $m-1$, and needs to be capable of resetting, to prepare for usage in the next block. To implement the counter, we add another block to Construction 1.

We start by describing a simple construction for a counter which we shall later append to Construction 1. Assume we have m flash memory cells indexed 0 to $m-1$, and with charge levels $c_0, \dots, c_{m-1}$. We shall say the counter encodes the integer $z \in \mathbb{Z}$, $z \geqslant 0$, if $0 \leqslant z \leqslant m-2$ is the smallest nonnegative integer for which $c_z > c_{z+1}$. If no such z exists, we shall say the counter encodes the value $m-1$. Using m cells the counter can assume the value of any integer in $\{0, \dots, m-1\}$. We note that when $1 \leqslant s < t$, in the (s,t,n)-LRM scheme every cell is comparable with its predecessor and successor. Thus, if the counter represents the value z, increasing it to $z+1$ involves a single push-to-the-top operation on cell $z+1$, ensuring both $c_z < c_{z+1}$ and $c_{z+1} > c_{z+2}$ (or just the former, if $z+1 = m-1$). Resetting the counter to represent a 0 is equally simple, and requires a single push-to-the-top operation on cell 0, ensuring $c_0 > c_1$.

Construction 2: The construction is a variation on Construction 1, so we shall describe it by noting the differences between them.

We set $n = m^2 + m$, and each codeword shall be made up of $m+1$ blocks of length m. The first m blocks are those described in Construction 1. Block m, the last block, will implement a counter. Thus, if g is a length m^2 codeword from Construction 1, and the counter represents the value z, we shall denote the codeword in the code we now construct as the pair (g, z).

Let $g_0, g_1, \dots, g_{L-1}$ be the anchor vectors from Construction 1, and assume ℓ_i is the number of auxiliary vectors between $g_{\ell-1}$ and g_ℓ (indices taken modulo L) in Construction 1. The new code we construct has anchors

$$g'_i = (g_i, m - \ell_i),$$

for all $i \in [L]$.

Finally, we create auxiliary vectors between adjacent anchors using Algorithm 2, which is a simple variation on the original Algorithm 1.

Algorithm 2 Transform Block k of Configuration $\mathbf{f_c}k$ from v_{s_i} to $v_{s_{i+m}}$ in a Fixed Number of Steps

```
Input: current cell configuration $\mathbf{f_c}$, block number $k$, new block value $v_{s_{i+m}}$
Output: new cell configuration $\mathbf{f_{c'}}$
Reset the counter.
Push cell $m-1$ (the last cell) in block $k-1$ (modulo $m$).
$a_j \Leftarrow 0$ for all $j = 0, 1, \ldots, m-3$
$z \Leftarrow 0$
$j \Leftarrow 0$
repeat
    if $v_{s_{i+m}}(j) = \sum_{i=j+1}^{r'(j)} a_i$ and $a_j = 0$ then
        if $c_{km+j} \leqslant \max_{l'(j) \leqslant j' \leqslant r'(j), a_{j'}=1} c_{km+j'}$ then
            Push the $j$th cell in block $k$ (the cell in position $km+j$).
            $z \Leftarrow z+1$
        end if
        $a_j \Leftarrow 1$
        $j \Leftarrow l'(j)$
    else
        $j \Leftarrow j+1$
    end if
until $j = m-2$
repeat
    Increment the counter.
    $z \Leftarrow z+1$
until $z = m$
Push cell $m-2$ (the next-to-last cell) in block $k$.
Output the resulting cell configuration $\mathbf{f_{c'}}$.
```

Intuitively, Algorithm 2 is the same as Algorithm 1 except for the counter reset at its beginning, and the counter increments at its end. These ensure the number of auxiliary vectors between anchors is constant. We observe the simple fact that anchor codewords have a nonzero counter, and so, when we reset the counter at the beginning of Algorithm 2 we obtain a different vector. Showing the codewords are distinct follows the exact same arguments as those used for Construction 1. In contrast with Construction 1, in Construction 2 we can give an exact expression for the size of the code, $L(m+2)$, since we have L anchors, and the distance between anchors is exactly $m+2$. It is also easy to show Corollary 9 also holds for a family of codes generated using Construction 2.

Finally, the next-state, encoding, and decoding algorithms may all be implemented efficiently for the codes from Construction 2. The next-state algorithm is essentially the same as that for the codes from Construction 1. As for encoding and decoding, the ith codeword may be uniquely described using $i_{\text{anchor}} = \lfloor i/(m+2) \rfloor$ which is the index of the nearest previous anchor in the code, and by $i_{\text{aux}} = i \bmod (m+2)$ which is the distance from the previous anchor.

Decoding is done by identifying the underlined block, and thus retrieving the correct position in the De-Bruijn sequence (see [17], [22] for example), which in turn, gives us i_{anchor}. If the codeword is not an anchor, we can run Algorithm 2 on the anchor until we reach the current codeword, and thus obtain i_{aux}. The decoded index is therefore $(m+2)i_{\text{anchor}} + i_{\text{aux}}$. Encoding is done in reverse, where we use an algorithm for encoding De-Bruijn sequences to find the appropriate De-Bruijn subsequence, translate it to the anchor of index i_{anchor}, and then run Algorithm 2 until i_{aux} push-to-the-top operations are made.

IV. Conclusion

We presented the framework for (s,t,n)-local rank modulation, and studied gray codes for the most general case. The codes we presented are asymptotically rate-optimal.

Several questions remain open. For the case of $(1,2,n)$-LRM, a previous work describes asymptotically rate-optimal codes for which the weight of the codewords is constant and approaches $\frac{n}{2}$[8]. That property guarantees a bounded charge difference in any “push-to-the-top” operation. Constant-weight codes for the general case are still missing. Of more general interest is the study of codes that cover a constant fraction of the space.

Acknowledgment

The authors would like to thank the Associate Editor and the anonymous reviewers, whose comments helped improve the presentation of this paper.

References

[1] T. Berger, F. Jelinek, and J. K. Wolf, “Permutation codes for sources,” *IEEE Trans. Inf. Theory*, vol. IT-18, no. 1, pp. 160–169, Jan. 1972.
[2] I. F. Blake, “Permutation codes for discrete channels,” *IEEE Trans. Inf. Theory*, vol. IT-20, no. 1, pp. 138–140, Jan. 1974.
[3] I. F. Blake, G. Cohen, and M. Deza, “Coding with permutations,” *Inf. Control*, vol. 43, pp. 1–19, 1979.
[4] H. Chadwick and I. Reed, “The equivalence of rank permutation codes to a new class of binary codes,” *IEEE Trans. Inf. Theory*, vol. IT-16, no. 5, pp. 640–641, Sep. 1970.
[5] H. D. Chadwick and L. Kurz, “Rank permutation group codes based on Kendall’s correlation statistic,” *IEEE Trans. Inf. Theory*, vol. IT-15, no. 2, pp. 306–315, Mar. 1969.
[6] G. Cohen and M. Deza, “Decoding of permutation codes,” presented at the Int. CNRS Colloquium, France, Jul. 1977.
[7] M. Deza and P. Frankl, “On maximal numbers of permutations with given maximal or minimal distance,” *J. Combin. Theory Ser. A*, vol. 22, pp. 352–360, 1977.
[8] E. E. Gad, M. Langberg, M. Schwartz, and J. Bruck, “Constant-weight gray codes for local rank modulation,” *IEEE Trans. Inf. Theory*, vol. 57, no. 11, pp. 7431–7442, Nov. 2011.
[9] H. C. Ferreira, A. J. H. Vinck, T. G. Swart, and I. de Beer, “Permutation trellis codes,” *IEEE Trans. Commun.*, vol. 53, no. 11, pp. 1782–1789, Nov. 2005.
[10] S. W. Golomb, *Shift Register Sequences*. San Francisco, CA, USA: Holden-Day, 1967.
[11] F. Gray, “Pulse code communication,” U.S. 2632058, Mar. 1953.
[12] A. Jiang, V. Bohossian, and J. Bruck, “Rewriting codes for joint information storage in flash memories,” *IEEE Trans. Inf. Theory*, vol. 56, no. 10, pp. 5300–5313, Oct. 2010.
[13] A. Jiang, M. Langberg, M. Schwartz, and J. Bruck, “Universal rewriting in constrained memories,” in *Proc. IEEE Int. Symp. Inf. Theory*, Seoul, Korea, Jun. 2009, pp. 1219–1223.
[14] A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck, “Rank modulation for flash memories,” *IEEE Trans. Inf. Theory*, vol. 55, no. 6, pp. 2659–2673, Jun. 2009.
[15] A. Jiang, M. Schwartz, and J. Bruck, “Correcting charge-constrained errors in the rank-modulation scheme,” *IEEE Trans. Inf. Theory*, vol. 56, no. 5, pp. 2112–2120, May 2010.

[16] C. A. Laisant, "Sur la numération factorielle, application aux permutations," *Bulletin de la Société Mathématique de France*, vol. 16, pp. 176–183, 1888.

[17] C. J. Mitchell, T. Etzion, and K. G. Paterson, "A method for constructing decodable De Bruijn sequences," *IEEE Trans. Inf. Theory*, vol. IT-42, no. 5, pp. 1472–1478, Sep. 1996.

[18] C. D. Savage, "A survey of combinatorial gray codes," *SIAM Rev.*, vol. 39, no. 4, pp. 605–629, Dec. 1997.

[19] R. Sedgewick, "Permutation generation methods," *Comput. Surv.*, vol. 9, no. 2, pp. 137–164, Jun. 1977.

[20] D. Slepian, "Permutation modulation," *Proc. IEEE*, vol. 53, no. 3, pp. 228–236, Mar. 1965.

[21] I. Tamo and M. Schwartz, "Correcting limited-magnitude errors in the rank-modulation scheme," *IEEE Trans. Inf. Theory*, vol. 56, no. 6, pp. 2551–2560, Jun. 2010.

[22] J. Tuliani, "De Bruijn sequences with efficient decoding algorithms," *Discrete Math.*, vol. 226, no. 1, pp. 313–336, Jan. 2001.

[23] H. Vinck, J. Haering, and T. Wadayama, "Coded M-fsk for power line communications," in *Proc. IEEE Int. Symp. Inf. Theory*, Sorrento, Italy, 2000, p. 137.

[24] Z. Wang, A. Jiang, and J. Bruck, "On the capacity of bounded rank modulation for flash memories," in *Proc. IEEE Int. Symp. Inf Theory*, Seoul, Korea, Jun. 2009, pp. 1234–1238.

[25] E. Yaakobi, P. H. Siegel, and J. K. Wolf, "Buffer codes for multi-level flash memory," presented at the IEEE Int. Symp. Inf. Theory, Toronto, ON, Canada, Jul. 2008.

[26] Y. Yehezkeally and M. Schwartz, "Snake-in-the-box codes for rank modulation," *IEEE Trans. Inf. Theory*, vol. 58, no. 8, pp. 5471–5483, Aug. 2012.

Eyal En Gad was born in Israel in 1982. He received the B.Sc. degree in 2008 from the Electrical Engineering Department, Technion—Israel Institute of Technology, Haifa, Israel. He is now a doctoral student with the Department of Electrical Engineering, California Institute of Technology, Pasadena. His research interests include information and coding theory with data storage applications.

Michael Langberg (M'07) is an Associate Professor in the Mathematics and Computer Science department at the Open University of Israel. Previously, between 2003 and 2006, he was a postdoctoral scholar in the Computer Science and Electrical Engineering departments at the California Institute of Technology. He received his B.Sc. in mathematics and computer science from Tel-Aviv University in 1996, and his M.Sc. and Ph.D. in computer science from the Weizmann Institute of Science in 1998 and 2003 respectively.

Dr. Langberg's research addresses the algorithmic and combinatorial aspects of information in communication, management, and storage. With special interest in information theory, coding theory, network communication and network coding.

Moshe Schwartz (M'03–SM'10) was born in Israel in 1975. He received the B.A. (summa cum laude), M.Sc., and Ph.D. degrees from the Technion—Israel Institute of Technology, Haifa, Israel, in 1997, 1998, and 2004 respectively, all from the Computer Science Department.

He was a Fulbright postdoctoral researcher in the Department of Electrical and Computer Engineering, University of California San Diego, and a postdoctoral researcher in the Department of Electrical Engineering, California Institute of Technology. He now holds a position with the Department of Electrical and Computer Engineering, Ben-Gurion University of the Negev, Israel. He is currently on a sabbatical as a Visiting Scientist in the Research Laboratory of Electronics, MIT, in Cambridge, MA.

Dr. Schwartz received the 2009 IEEE Communications Society Best Paper Award in Signal Processing and Coding for Data Storage, and the 2010 IEEE Communications Society Best Student Paper Award in Signal Processing and Coding for Data Storage. His research interests include algebraic coding, combinatorial structures, and digital sequences.

Jehoshua Bruck (S'86–M'89–SM'93–F'01) received the B.Sc. and M.Sc. degrees in electrical engineering from the Technion-Israel Institute of Technology, Haifa, Israel, in 1982 and 1985, respectively, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 1989. He is the Gordon and Betty Moore Professor of computation and neural systems and electrical engineering at the California Institute of Technology, Pasadena, CA. His extensive industrial experience includes working with IBM Almaden Research Center, as well as cofounding and serving as Chairman of Rainfinity, acquired by EMC in 2005; and XtremIO, acquired by EMC in 2012. His current research interests include information theory and systems and the theory of computation in biological networks.

Dr. Bruck is a recipient of the Feynman Prize for Excellence in Teaching, the Sloan Research Fellowship, the National Science Foundation Young Investigator Award, the IBM Outstanding Innovation Award, and the IBM Outstanding Technical Achievement Award. His papers were recognized in journals and conferences, including winning the 2010 IEEE Communications Society Best Student Paper Award in Signal Processing and Coding for Data Storage for his paper on codes for limited-magnitude errors in flash memories, the 2009 IEEE Communications Society Best Paper Award in Signal Processing and Coding for Data Storage for his paper on rank modulation for flash memories, the 2005 A. Schelkunoff Transactions Prize Paper Award from the IEEE Antennas and Propagation Society for his paper on signal propagation in wireless networks, and the 2003 Best Paper Award in the 2003 Design Automation Conference for his paper on cyclic combinational circuits.