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 multi-level 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 our constructed 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], [13], [14], [18], [21]. In the conventional modulation scheme used in flash-memory cells, the absolute charge level of each cell is quantized to one of q 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, then no over-programming can occur, a problem which considerably slows down programming in conventional multi-level flash cells.

Rank modulation has been studied intermittently since the early works of Slepian [17] (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 [20]), and for modulation schemes for flash memory [13], [14], [18], [21].

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], [21] 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 one-dimensional 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,

$c_0 = 5.00 \quad c_1 = 2.50 \quad c_2 = 4.25 \quad c_3 = 6.50 \quad c_4 = 4.00 \quad c_5 = 1.00 \quad c_6 = 1.50 \quad c_7 = 5.50 \quad c_8 = 6.00$

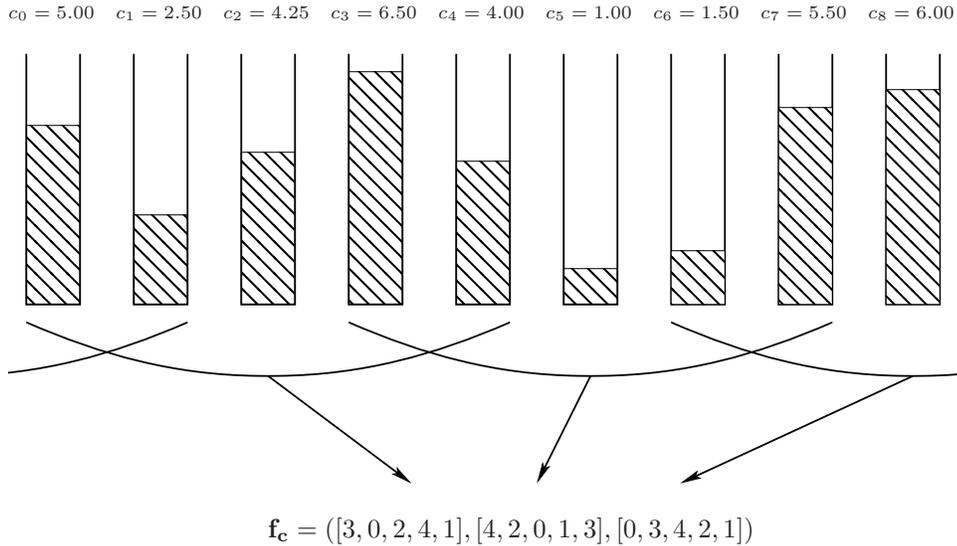$$\mathbf{f_c} = ([3, 0, 2, 4, 1], [4, 2, 0, 1, 3], [0, 3, 4, 2, 1])$$

Fig. 1. Demodulating a $(3, 5, 9)$-locally rank-modulated signal.

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 [13]: A set of n cells, over which the rank-modulation scheme is applied, is used to simulate a single conventional multi-level flash cell with $n!$ levels corresponding to the alphabet $\{0, 1, \ldots, n! - 1\}$. 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 [11], [12], [22]), and it is realized in [13] 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. In the context of local rank modulation, Gray codes for the local scheme were studied in [8], where necessary conditions as well as constructions were provided.

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 2 elements, the question of whether any middle-road solutions exist remains open. We address this question in this paper by considering the generalized local rank modulation scheme in which a sequence of several permutations of a given size provide the local views into 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 our 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

Let us consider a sequence of t real-valued variables, $\mathbf{c} = (c_0, c_1, \ldots, c_{t-1}) \in \mathbb{R}^t$, where 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 $[t] = \{0, 1, 2, \ldots, t - 1\}$. The permutation $f_{\mathbf{c}}$ is defined as

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

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

Given a sequence of n variables, $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$, we define a window of size t at position p to be

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

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, $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$, the demodulation maps $\mathbf{c}$ to the sequence of n/s permutations from S_t as follows:

$$\mathbf{f_c} = (f_{\mathbf{c}_{0,t}}, f_{\mathbf{c}_{s,t}}, f_{\mathbf{c}_{2s,t}}, \ldots, f_{\mathbf{c}_{n-s,t}}). \qquad (1)$$

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, $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$, to be the charge-level readings from n flash cells. The demodulated sequence, $\mathbf{f_c}$, 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], [13], [14], [18], [21]. See Figure 1 for an example.

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 $\mathbf{f} = \mathbf{f_c}$, 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. 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 $(t \cdot (t-1) \cdot \ldots \cdot (t-s+1))^{n/s}$ (this fact is also stated in [21]).

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 [15] for the earliest-known definition, and [13] for a related use): We can represent any permutation $f = [f(0), \ldots, f(t-1)] \in S_t$ with a sequence of digits $d_{t-1}, d_{t-2}, \ldots, d_1, d_0$, where $d_i \in \mathbb{Z}_i$, and d_i counts the number of elements to the right of $f(i)$ which are of lower value, i.e., $d_i = |\{j > i \mid c_j < c_i\}|$. 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 our succinct representation, we now contend that due to the overlap between local views, we can then 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 $\bar{f}_{\mathbf{c}_{i \cdot s,t}}$. Accordingly, we define,

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

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

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: That $\left| \bar{\mathcal{R}}(s,t,n) \right| \leqslant |\mathcal{R}(s,t,n)|$ is trivial, since any $\mathbf{f} \in \mathcal{R}(s,t,n)$ results in one $\bar{\mathbf{f}} \in \bar{\mathcal{R}}(s,t,n)$. For the other inequality, 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 $t!/(t-s)!$ ways of choosing $f_{\mathbf{c}_{n-s,t}}$, and then the same amount of ways of choosing $f_{\mathbf{c}_{n-2s,t}}$, and continuing all the way up to $f_{\mathbf{c}_{0,t}}$ we get the desired result. ∎

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.* [21] (though in a slightly different manner where indices are not taken modulo n, i.e., with no wrap-around). In [21], 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

Generally speaking, a *Gray code*, G, is a sequence of distinct states (codewords), $G = g_0, g_1, \ldots, g_{N-1}$, 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, our 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 [13], 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. 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 j-th cell. More precisely, let $\mathbf{f}$ be an (s,t,n)-LRM realizable sequence of permutations, i.e., there exists $\mathbf{c} \in \mathbb{R}^n$ such that $\mathbf{f} = \mathbf{f_c}$. Now define the transition τ_j acting on $\mathbf{f}$ as $\mathbf{f}' = \mathbf{f}'_{\mathbf{c}'}$ realizable by the variables $\mathbf{c}' = (c'_0, \ldots, c'_{n-1}) \in \mathbb{R}^n$ such that c'_j is pushed to a value higher than all of it's comparable cells. We denote $r(j)$ as the rightmost index (cyclically) among the cells that share a window with c'_j, and $l(j)$ as the leftmost index (cyclically) among them. We can find $r(j)$ and $l(j)$ by the following expressions:

$$l(j) = s \left\lceil \frac{j-t+1}{s} \right\rceil \mod n,$$

$$r(j) = \left(s \left\lfloor \frac{j}{s} \right\rfloor + (t-1) \right) \mod n.$$

Now $\mathbf{c}'$ is given by the following expression:

$$c'_i = \begin{cases} c_i & i \neq j, \\ \max \left\{ c_{l(i)}, \ldots, c_{r(i)} \right\} + 1 & i = j. \end{cases}$$

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. *We say 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 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-optimal* Gray codes for

(s,t,n)-LRM. A rough description of our construction follows. First we partition the n cells into m blocks each of size m/n. To simplify our presentation we set $m = \sqrt{n}$, 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 in $\bar{\mathcal{R}}(s,t,m)$. Namely, each and every block can be thought of an element of an alphabet $\Sigma = \{v_0, \ldots, v_{V-1}\}$ of size 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, sub-indices of s are taken modulo V^{m-1}. Here, $s_i \in [V]$. Such sequences S exist, e.g., [10].

We are now ready to construct our Gray code G. The construction will have two phases. First we construct 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:

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

where $L = \text{lcm}(m, V^{m-1})$, the sub-indices 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 $\bar{G}$ does not suffice for our construction as the transitions between the anchors g_i and g_{i+1} involve 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. Our construction is summarized below formally.

As will be seen in our analysis, it is crucial that we may identify for each each element in $\bar{G}$ exactly which block is being changed. To this end, and for other technical reasons, we will add some auxiliary digits to each block. These digits are referred to as *non-information* digits. Loosely speaking, in each block of size m the last $t+1$ digits (approximately[1]) will be non information digits. The last 2 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 $t+1$ non-information digits will be negligible with respect to the remaining $m - (t+1)$ information digits, allowing our codes to have asymptotically optimal rate.

Construction 1. *We consider the (s,t,n)-LRM, where n is a square, $m = \sqrt{n} \geqslant t+2$, and $s|m$. Let $\{v_0, v_1, \ldots, v_{V-1}\}$ be a set of V distinct mixed-radix vectors of length m taken from $([t] \times [t-1] \times \cdots \times [t-s])^{m/s}$. The values of the last $s(\lceil (t+2)/s \rceil - 1)$ digits of each v_i do not play a role in the representation of the stored data and are called* non-information digits*, so by abuse of notation, a mixed-radix vector $(r_0, r_1, \ldots, r_{m-1})$ actually represents the value $(r_0, r_1, \ldots, r_{m-1-s(\lceil (t+2)/s \rceil - 1)})$ regardless of the value of the last $s(\lceil (t+2)/s \rceil - 1)$ elements. Therefore, we get*

$$V = \left(\frac{t!}{(t-s)!} \right)^{\frac{m}{s} - \lceil \frac{t+2}{s} \rceil + 1}.$$

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

Consider a de-Bruijn sequence S of order $m-1$ over the alphabet $\{0, 1, \ldots, V-1\}$. The Gray code $\bar{G}$ of anchor vectors is a sequence $g_0, g_1, \ldots, g_{L-1}$ of L mixed-radix vectors of length $m^2 = n$. Each vector is formed by a concatenation of m blocks of length m. Between the anchors g_i and g_{i+1}, the block v_{s_i} is transformed into the block $v_{s_{i+m}}$.

Within each of the m blocks comprising any single anchor, the $(m-2)$nd digit (the second-from-right digit – a non-information digit) is set to its maximal value in all blocks except for the "underlined" block (i.e., the block being currently changed). For brevity, we call this digit the underline digit*.*

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.*

[1]for the exact value see construction 1

In what follows we present Algorithm 1 that specifies the sequence $g_i^0, g_i^1, \ldots, g_i^{\ell_i}$ that allow 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 v_{s_i} to $v_{s_{i+m}}$), the same will hold for the sequence $g_i^0, g_i^1, \ldots, g_i^{\ell_i}$, i.e., g_i^j and $g_i^{j'}$ will only differ 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, for administrative reasons, in the block to the left (in cyclic order), we also push cells.

We now present Algorithm 1 and describe some of its properties. We then prove that indeed the resulting code G is an asymptotically-optimal cyclic (s,t,n)-LRMGC. We assume that the following algorithm is applied to positions $\{0, 1, \ldots, m-1\}$. We further assume $(r_0, r_1, \ldots, r_{m-1}) \in ([t] \times [t-1] \times \cdots \times [t-s])^{m/s}$ represents the value v_ℓ, then we say the jth digit of v_ℓ is

$$v_\ell(j) = \begin{cases} r_j & 0 \leqslant j < m - s(\lceil (t+2)/s \rceil - 1) \\ 0 & \text{otherwise.} \end{cases}$$

Finally, we restrict $l(\cdot)$ and $r(\cdot)$ by defining

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

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

Our algorithm changes a block of length m with cell values that represent v_{s_i} into one that represents $v_{s_{i+m}}$ using push-to-the-top operations. It 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 to the right of j and are of value lower than c_j. Roughly speaking, to obtain such a configuration $\mathbf{c}$, our algorithm, for $j \in [m]$, *marks* each cell c_j in $\mathbf{c}$ after exactly $v_{s_{i+m}}(j)$ cells to the right of c_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.

Algorithm 1 Transform block v_{s_i} to block $v_{s_{i+m}}$

In the block to the left (in cyclic order), push the rightmost cell

$a_j \Leftarrow 0$ for all $j = 0, 1, \ldots, m-3$

$j \Leftarrow 0$

repeat

  if $v_{s_{i+m}}(j) = \sum\limits_{i=j+1}^{r'(j)} a_i$ and $a_j = 0$ **then**

    if $c_j \leqslant \max\limits_{l'(j) \leqslant k \leqslant r'(j), a_k = 1} c_k$ **then**

      Push the jth cell of current block.

    end if

    $a_j \Leftarrow 1$

    $j \Leftarrow l'(j)$

  else

    $j \Leftarrow j + 1$

  end if

until $j = m - 2$

Push the next-to-last cell of current block.

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. As each time a cell is changed it is pushed to the top, this will ensure that the resulting cell configuration **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.

Example 1. *Take the case of* $(1, 2, 16)$-*LRM with* $m = 4$, $V = 2$, *and a de-Bruijn sequence* $S = 00010111$ *of order* 3 *and alphabet of size* 2. *The list of anchors is*

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

The bold bit (the leftmost bit in each group of four) denotes the information bit, while the rest are non-information bits. The underlined vectors are easily recognizable by the next-to-rightmost 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 *be 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)*

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

We now address the analysis of Algorithm 1.

Lemma 2. *Assuming the underline 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 underline digit in anchors. In addition, between any two adjacent anchors, Algorithm* 1 *guarantees the underline digits of the changing block and the block to its left (cyclically), are both not of maximal value (amoung the cells with which they share a window).*

Proof: The proof is by induction. The base case follows from our 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 rightmost cell of the block to the left of that being changed. This implies that the value of the underline cell in both the block being changed and that to its left 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. At that point in time, the underline cell in the changed block obtains its maximal value, while the block to its left (that to be changed in the next application of Algorithm 1) is of non-maximal value. All the other underline cells remain unchanged throughout the execution of Algorithm 1. ∎

Lemma 4. *Every word in the Gray code is distinct.*

Proof: First, by lemmas 2 and 3, we know that all of the anchors in our construction are distinct. It is left to show that adding the auxiliary vectors resulting from Algorithm 1, our code remains a Gray code. Here 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 k such that $c_j < c_k$ and $a_k = 1$. After the push, $c_j > c_k$. But since $a_k = 1$, cell k will not be pushed anymore, and thus in all future configurations cell j will be higher than cell k. 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: That Algorithm 1 terminates is easy to see. For convenience, we denote $z = s(\lceil(t+2)/s\rceil - 1)$. For each $k \in \{m-z,\ldots,m-3\}$, $v_{s_{i+m}}(k) = 0$, and therefore each of those cells is marked the first time that $j = k$. Now we assume by induction that for each $k \leqslant m-z$, all of the cells with indices l, $k \leqslant l \leqslant m-3$, are marked before the algorithm terminates.

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

Theorem 1. *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 $\{j+1,\ldots,r'(j)\}$ 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 6. *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 our observations up to now, we are able to summarize with the following theorem for G from Construction 1.

Theorem 2. *The code G from Construction* 1 *is a cyclic Gray code of size at least L.*

We note also that G can be encoded and decoded efficiently. That is true since algorithm 1 is efficient, and since there exist efficient encoding and decoding procedures for de-Bruijn sequences, such as the ones described in [16], [19].

Corollary 1. *For all constants $1 \leqslant s < t$, there exists an asymptotically-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 $i \geqslant t+2$. Then $N_i \geqslant L_i \geqslant V_i^{si-1}$. It follows that

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

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 present are asymptotically optimal.

Several questions remain open. For the case of $(1,2,n)$-LRM, a previous work describes asymptotically-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.

REFERENCES

[1] T. Berger, F. Jelinek, and J. K. Wolf, "Permutation codes for sources," *IEEE Trans. on Inform. Theory*, vol. IT-18, no. 1, pp. 160–169, Jan. 1972.

[2] I. F. Blake, "Permutation codes for discrete channels," *IEEE Trans. on Inform. Theory*, vol. 20, pp. 138–140, 1974.

[3] I. F. Blake, G. Cohen, and M. Deza, "Coding with permutations," *Inform. and 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. on Inform. Theory*, vol. 16, no. 5, pp. 640–641, 1970.

[5] H. D. Chadwick and L. Kurz, "Rank permutation group codes based on Kendall's correlation statistic," *IEEE Trans. on Inform. Theory*, vol. IT-15, no. 2, pp. 306–315, Mar. 1969.

[6] G. Cohen and M. Deza, "Decoding of permutation codes," in *Intl. CNRS Colloquium, July, France*, 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, 1977.

[8] E. En Gad, M. Langberg, M. Schwartz, and J. Bruck, "Constant-weight gray codes for local rank modulation," *IEEE Trans. on Inform. Theory*, vol. 57, no. 11, pp. 7431–7442, Nov. 2011.

[9] H. C. Ferriera, A. J. H. Vinck, T. G. Swart, and I. de Beer, "Permutation trellis codes," *IEEE Trans. on Communications*, pp. 1782–1789, Nov. 2005.

[10] S. W. Golomb, *Shift Register Sequences*. Holden-Day, San Francisco, 1967.

[11] A. Jiang, V. Bohossian, and J. Bruck, "Rewriting codes for joint information storage in flash memories," *IEEE Trans. on Inform. Theory*, vol. 56, no. 10, pp. 5300–5313, Oct. 2010.

[12] A. Jiang, M. Langberg, M. Schwartz, and J. Bruck, "Universal rewriting in constrained memories," in *Proceedings of the 2009 IEEE International Symposium on Information Theory (ISIT2009), Seoul, Korea*, Jun. 2009, pp. 1219–1223.

[13] A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck, "Rank modulation for flash memories," *IEEE Trans. on Inform. Theory*, vol. 55, no. 6, pp. 2659–2673, Jun. 2009.

[14] A. Jiang, M. Schwartz, and J. Bruck, "Correcting charge-constrained errors in the rank-modulation scheme," *IEEE Trans. on Inform. Theory*, vol. 56, no. 5, pp. 2112–2120, May 2010.

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

[16] C. J. Mitchell, T. Etzion, and K. G. Paterson, "A method for constructing decodable de Bruijn sequences," *IEEE Trans. on Inform. Theory*, vol. IT-42, pp. 1472–1478, 1996.

[17] D. Slepian, "Permutation modulation," in *Proc. of the IEEE*, vol. 53, no. 3, 1965, pp. 228–236.

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

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

[20] H. Vinck, J. Haering, and T. Wadayama, "Coded M-FSK for power line communications," in *Proceedings of the 2000 IEEE International Symposium on Information Theory (ISIT2000), Sorrento, Italy*, 2000, p. 137.

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

[22] E. Yaakobi, P. H. Siegel, and J. K. Wolf, "Buffer codes for multi-level flash memory," in *Proceedings of the 2008 IEEE International Symposium on Information Theory (ISIT2008), Toronto, Canada*, 2008, poster.