2019 57th Annual Allerton Conference on Communication, Control, and Computing
(Allerton)
Allerton Park and Retreat Center
Monticello, IL, USA, September 24-27, 2019

Asymmetric LOCO Codes: Constrained Codes for Flash Memories

Ahmed Hareedy and Robert Calderbank
Electrical and Computer Engineering Department, Duke University, Durham, NC 27705 USA
ahmed.hareedy@duke.edu and robert.calderbank@duke.edu

***Abstract*—In data storage and data transmission, certain patterns are more likely to be subject to error when written (transmitted) onto the media. In magnetic recording systems with binary data and bipolar non-return-to-zero signaling, patterns that have insufficient separation between consecutive transitions exacerbate inter-symbol interference. Constrained codes are used to eliminate such error-prone patterns. A recent example is a new family of capacity-achieving constrained codes, named lexicographically-ordered constrained codes (LOCO codes). LOCO codes are symmetric, that is, the set of forbidden patterns is closed under taking pattern complements. LOCO codes are suboptimal in terms of rate when used in Flash devices where block erasure is employed since the complement of an error-prone pattern is not detrimental in these devices. This paper introduces asymmetric LOCO codes (A-LOCO codes), which are lexicographically-ordered constrained codes that forbid only those patterns that are detrimental for Flash performance. A-LOCO codes are also capacity-achieving, and at finite-lengths, they offer higher rates than the available state-of-the-art constrained codes designed for the same goal. The mapping-demapping between the index and the codeword in A-LOCO codes allows low-complexity encoding and decoding algorithms that are simpler than their LOCO counterparts.**

I. Introduction

Constrained codes are widely applied in various data storage systems to improve their performance. These codes were first employed in magnetic recording (MR) systems. Run-length-limited (RLL) codes [1] were used to extend the lifetime of peak detection in early IBM disk drives [2]. Binary RLL codes are typically used with bipolar non-return-to-zero inverted (NRZI) signaling, where a 0 is represented by no transition, while a 1 is represented by a transition. RLL codes are used to control the separation between consecutive transitions. Small separation exacerbates inter-symbol interference (ISI) while large separation results in losing synchronization at the receiver. Constrained codes also find application in modern MR systems [3] to improve sequence detection [4].

We define the set $\mathcal{S}_x \triangleq \{010, 101, 0110, 1001, \ldots, 0\mathbf{1}^x0, 1\mathbf{0}^x1\}$, where we denote a run of r consecutive 0's (resp., 1's) as $\mathbf{0}^r$ (resp. $\mathbf{1}^r$). Note that the set $\mathcal{S}_x$ is closed under taking pattern complements and has size $2x$. We define a binary symmetric $\mathcal{S}_x$-constrained code to be a code that forbids any pattern in the set $\mathcal{S}_x$ from appearing in any of its codewords. $\mathcal{S}_x$-constrained codes are used with bipolar non-return-to-zero (NRZ) signaling to control the separation between consecutive transitions. In NRZ signaling, a 0 is represented by level $-A$ or the erasure level E in Flash, while a 1 is represented by level $+A$. Many applications, e.g., optical recording, require constrained codes to be balanced [5].

In Flash systems employing block erasure, the error-prone patterns, i.e., the patterns that contribute the most to inter-cell interference (ICI), for NRZ signaling are somewhat different. It was demonstrated in [6] that for Flash memories, the pattern $(q-1)0(q-1)$ should be eliminated, where q is the number of levels in the cell and also the Galois field (GF) size[1]. Balanced and constant-weight codes were designed to eliminate this pattern in [6] and [7], respectively. The authors of [8] showed that the set of patterns to eliminate in multi-level cell (MLC) Flash memories is $\{303, 313, 323\}$, which can be generalized to $\{(q-1)0(q-1), (q-1)1(q-1), \ldots, (q-1)(q-2)(q-1)\}$ for q-level cell Flash memories as shown in [9]. The problem originates from the phenomenon that increasing the charge at the outer two cells causes the middle cell to have its charge also increased unintentionally because of the parasitic capacitances. Prior work only considered adjacent cells [6]–[9]. However, parasitic capacitances may also result in charge propagation between non-adjacent cells, which means patterns like $(q-1)\mathbf{0}^x(q-1)$, $x > 1$, can also be problematic (investigated here in the binary case).

We define the set $\mathcal{A}_x \triangleq \{101, 1001, \ldots, 1\mathbf{0}^x1\}$ and note that $\mathcal{A}_x$ has size x. We define a binary asymmetric $\mathcal{A}_x$-constrained code to be a code that forbids any pattern in the set $\mathcal{A}_x$ from appearing in any of its codewords. The code is said to be asymmetric because $\mathcal{A}_x$ is not closed under taking pattern complements. Here, NRZ signaling is adopted. For magnetic recording channels having the extended partial-response 4 (EPR4) target, the authors of [10] showed that asymmetric $\mathcal{A}_1$-constrained codes (forbidding $\{101\}$) achieve the same performance as symmetric $\mathcal{S}_1$-constrained codes (forbidding $\{010, 101\}$) with 20% rate increase.

The idea of constructing constrained codes based on lexicographic indexing (also called enumerative coding) was first presented in [1] for RLL codes. The framework introduced in [11] inspired more recent developments such as [12] and [13]. These developments have the drawback that the code length needs to be large for rates approaching capacity, resulting in limited rate-complexity trade-off advantages. The asymmetric constrained codes in [9], designed for Flash systems, are enumerative constant-composition codes with high rates and average encoding-decoding complexity. However, these codes are limited in the sense that only the effect of the two adjacent cells is taken into account, i.e., for a single-level cell (SLC) Flash memory device (inaccurate nomenclature; it is a single-bit cell), they are only $\mathcal{A}_1$-constrained codes.

[1]We map GF elements to integers representing threshold voltage levels.

978-1-7281-3151-1/19/$31.00 ©2019 IEEE

Recently, we introduced a new family of symmetric constrained codes, which we named lexicographically-ordered $\mathcal{S}_x$-constrained codes (LOCO codes) [14]. LOCO codes are capacity-achieving, and they offer up to 10% rate gain, with low complexity encoding-decoding, compared with practical RLL codes designed for the same goal. A combination of LOCO codes and spatially-coupled graph-based codes [15] resulted in significant density gains with limited rate reduction. Balancing LOCO codes was also proved to result in the minimum penalty in code rate. See [14] for details.

In this paper, we propose and analyze a new family of asymmetric constrained codes, which we name asymmetric lexicographically-ordered $\mathcal{A}_x$-constrained codes (A-LOCO codes), that improve performance by eliminating the error-prone patterns in Flash memories. Our A-LOCO codes can be constructed, encoded, and decoded for any set $\mathcal{A}_x$, making them capable of taking into account the effect of non-adjacent cells when needed. A-LOCO codes are capacity-achieving codes, and we establish a mapping-demapping formula between the lexicographic index and the codeword in order to enable simple, practical encoding and decoding algorithms. Compared with other practical asymmetric and symmetric constrained codes designed for the same purpose, A-LOCO codes offer higher rates at low complexity. In this paper, we only consider binary asymmetric constrained codes for SLC Flash memories. However, we expect to be able to develop non-binary asymmetric constrained codes for Flash memories with $q = 2^y$, $y \geq 2$, levels. High rate non-binary asymmetric constrained codes will encourage the development of quad-level cell (QLC) Flash memories.

The rest of the paper is organized as follows. In Section II, we enumerate the codewords in an A-LOCO code. In Section III, we establish the encoding-decoding rule of A-LOCO codes. In Section IV, we discuss bridging, self-clocking, and rates of A-LOCO codes. In Section V, we introduce the algorithms, complexity analysis, and comparisons with other constrained codes. We conclude the paper in Section VI.

II. Cardinality of A-LOCO Codes

In this section, we formally define A-LOCO codes, and then derive a recursive relation that gives the cardinality, i.e., the number of codewords, of these codes.

Definition 1. *An A-LOCO code $\mathcal{AC}_{m,x}$, with parameters $m \geq 1$ and $x \geq 1$, is defined via the following properties:*

1) *Codewords in $\mathcal{AC}_{m,x}$ are binary and of length m.*
2) *Codewords in $\mathcal{AC}_{m,x}$ are ordered lexicographically.*
3) *Any pattern in the asymmetric set $\mathcal{A}_x$ does not appear in any codeword $\mathbf{c}$ in $\mathcal{AC}_{m,x}$, where:*

$$\mathcal{A}_x \triangleq \{101, 1001, \ldots, 1\mathbf{0}^x1\}. \tag{1}$$

4) *The code $\mathcal{AC}_{m,x}$ contains all the codewords satisfying the previous three properties.*

Lexicographic ordering of codewords means that they are ordered in an ascending manner following the rule $0 < 1$ for any bit, and the bit significance reduces from left to right.

Our main application in this work is Flash memories. In SLC devices, a 1 results in a programmed cell, while a 0 results in an unprogrammed cell, i.e., NRZ signaling. Thus, patterns of the form $1\mathbf{0}^x1$ are error-prone since they give rise to ICI on the inner cell(s) (the unprogrammed cell(s)).

Table I shows the codewords of the A-LOCO codes $\mathcal{AC}_{m,1}$, $m \in \{1, 2, \ldots, 5\}$. The table demonstrates that an A-LOCO code with $m \notin \{1, 2\}$ is not closed under taking codeword complements. Moreover, the table also exhibits the increase in the A-LOCO code cardinality compared with the corresponding LOCO code. For example, the cardinality of the A-LOCO code with $m = 5$ and $x = 1$ is 21, while it is only 16 for the corresponding LOCO code [14].

Next, we introduce a group structure for A-LOCO codes that helps us not only derive the cardinality recursively, but also devise the encoding-decoding rule of A-LOCO codes, which is based on lexicographic indexing.

For $m \geq 2$, the codewords in an A-LOCO code $\mathcal{AC}_{m,1}$ are classified into the following three groups:

Group 1: Codewords that start with 0 from the left, i.e., at the left-most bit (LMB).

Group 2: Codewords that start with 11 from the left.

Group 3: Codewords that start with $1\mathbf{0}^{x+1}$ from the left.

This group structure is shown explicitly in Table I for $\mathcal{AC}_{5,1}$. Additionally, the horizontal lines in each column of codewords separate different groups. Note that bridging bits/symbols are required in order to guarantee that the forbidden patterns do not appear in streams of consecutive A-LOCO codewords. Bridging will be discussed later.

Theorem 1 derives the cardinality of A-LOCO codes.

Theorem 1. *Denote the cardinality of an A-LOCO code $\mathcal{AC}_{m,x}$ by $N(m, x)$ with:*

$$N(m, x) \triangleq 1, \ m \leq 0, \ \textit{and} \ N(1, x) \triangleq 2. \tag{2}$$

The following recursive equation gives $N(m, x)$:

$$\begin{aligned} N(m, x) = {} & 2N(m-1, x) - N(m-2, x) \\ & + N(m-x-2, x), \ m \geq 2. \end{aligned} \tag{3}$$

Proof: We use the group structure illustrated above in order to prove Theorem 1.

Group 1: Each codeword from Group 1 in $\mathcal{AC}_{m,x}$ corresponds to a codeword in $\mathcal{AC}_{m-1,x}$ that shares the $m-1$ right-most bits (RMBs) with the codeword in $\mathcal{AC}_{m,x}$. This applies to all the codewords in $\mathcal{AC}_{m-1,x}$. Recall that patterns of the form 01^y0, $1 \leq y \leq x$, are not forbidden in A-LOCO codes. Thus, the cardinality of Group 1 in $\mathcal{AC}_{m,x}$ is:

$$N_1(m, x) = N(m-1, x). \tag{4}$$

Group 2: Each codeword from Group 2 in $\mathcal{AC}_{m,x}$ corresponds to a codeword in $\mathcal{AC}_{m-1,x}$ that starts with 1 from the left and shares the $m-2$ RMBs with the codeword in $\mathcal{AC}_{m,x}$. This applies to all the codewords starting with 1 from the left in $\mathcal{AC}_{m-1,x}$. Thus, the cardinality of Group 2 in $\mathcal{AC}_{m,x}$ is:

$$N_2(m, x) = N(m-1, x) - N_1(m-1, x). \tag{5}$$

Using (4) to compute $N_1(m-1, x)$ gives:

TABLE I

THE CODEWORDS OF FIVE A-LOCO CODES, $\mathcal{AC}_{m,1}$, $m \in \{1, 2, \ldots, 5\}$ AND $x = 1$. THE THREE DIFFERENT GROUPS OF CODEWORDS ARE SHOWN FOR THE CODE $\mathcal{AC}_{5,1}$.

Codeword index $g(\mathbf{c})$	Codewords of the code $\mathcal{AC}_{m,1}$					
	$m=1$	$m=2$	$m=3$	$m=4$	$m=5$	
0	0	00	000	0000	00000	
1	1	01	001	0001	00001	
2		10	010	0010	00010	
3		11	011	0011	00011	
4			100	0100	00100	
5			110	0110	00110	Group 1
6			111	0111	00111	
7				1000	01000	
8				1001	01001	
9				1100	01100	
10				1110	01110	
11				1111	01111	
12					10000	
13					10001	Group 3
14					10010	
15					10011	
16					11000	
17					11001	
18					11100	Group 2
19					11110	
20					11111	
Code cardinality	$N(1,1) \triangleq 2$	$N(2,1) = 4$	$N(3,1) = 7$	$N(4,1) = 12$	$N(5,1) = 21$	

$$N_2(m,x) = N(m-1,x) - N(m-2,x). \tag{6}$$

Group 3: Each codeword from Group 3 in $\mathcal{AC}_{m,x}$ corresponds to a codeword in $\mathcal{AC}_{m-x-1,x}$ that starts with 0 from the left and shares the $m-x-2$ RMBs with the codeword in $\mathcal{AC}_{m,x}$. This applies to all the codewords starting with 0 from the left in $\mathcal{AC}_{m-x-1,x}$. Thus, the cardinality of Group 3 in $\mathcal{AC}_{m,x}$ is:

$$N_3(m,x) = N_1(m-x-1,x). \tag{7}$$

Using (4) to compute $N_1(m-x-1,x)$ gives:

$$N_3(m,x) = N(m-x-2,x). \tag{8}$$

Adding (4), (6), and (8) gives:

$$\begin{aligned} N(m,x) &= \sum_{\ell=1}^{3} N_\ell(m,x) \\ &= 2N(m-1,x) - N(m-2,x) + N(m-x-2,x), \end{aligned}$$

which completes the proof. ∎

Example 1. *The cardinalities of $\mathcal{AC}_{m,1}$, $m \in \{2,3,4,5\}$, are computed using Theorem 1 as follows:*

$$\begin{aligned} &N(-1,1) \triangleq 1,\ N(0,1) \triangleq 1,\ N(1,1) \triangleq 2, \\ &N(2,1) = 2N(1,1) - N(0,1) + N(-1,1) = 4, \\ &N(3,1) = 2N(2,1) - N(1,1) + N(0,1) = 7, \\ &N(4,1) = 2N(3,1) - N(2,1) + N(1,1) = 12, \\ &N(5,1) = 2N(4,1) - N(3,1) + N(2,1) = 21, \end{aligned}$$

which are also given in the last row of Table I.

Theorem 1 is important because, for a given length m, the number of codewords determines the rate of the code. This is true for all coding techniques based on lexicographic ordering (true for enumerative coding techniques in general). Theorem 1 is also essential for devising the encoding-decoding rule of A-LOCO codes as we shall see next section, and consequently, the encoding-decoding algorithms.

III. ENCODING-DECODING RULE OF A-LOCO CODES

In this section, we devise the encoding-decoding rule of A-LOCO codes, which is based on lexicographic indexing. This rule is what enables simple, low complexity encoding and decoding algorithms for A-LOCO codes.

We first introduce some notation. Define an A-LOCO codeword of length m as $\mathbf{c} \triangleq [c_{m-1}\ c_{m-2}\ \ldots\ c_0] \in \mathcal{AC}_{m,x}$. The index of an A-LOCO codeword $\mathbf{c}$ in $\mathcal{AC}_{m,x}$ is denoted by $g(m,x,\mathbf{c})$, which is sometimes abbreviated to $g(\mathbf{c})$, as in Table I, for simplicity. We also define an integer variable a_i for each binary c_i as follows:

$$a_i \triangleq \begin{cases} 1, & c_i = 1, \\ 0, & c_i = 0, \end{cases} \tag{9}$$

with $a_m \triangleq 0$. The same notation applies for an A-LOCO codeword of length $m+1$, $\mathbf{c}'$ in $\mathcal{AC}_{m+1,x}$, and an A-LOCO codeword of length $m-x$, $\mathbf{c}''$ in $\mathcal{AC}_{m-x,x}$.

Theorem 2 gives the encoding-decoding rule of A-LOCO codes. The indexing is trivial for the case of $m = 1$

Theorem 2. *The lexicographic index $g(m,x,\mathbf{c})$ of an A-LOCO codeword $\mathbf{c}$ in $\mathcal{AC}_{m,x}$, $m \geq 2$ and $x \geq 1$, is computed from the codeword itself according to the following rule:*

$$g(\mathbf{c}) = \sum_{i=0}^{m-1} a_i N(i - a_{i+1}x, x). \tag{10}$$

Proof: We prove Theorem 2 by induction.

Base: Our base case is the case of $m = 2$. For $\mathcal{AC}_{2,x}$, we always have four codewords, say $\mathbf{c}_0$, $\mathbf{c}_1$, $\mathbf{c}_2$, and $\mathbf{c}_3$ in order, for any value of x. These four codewords are listed in Table I. We want to prove that $g(\mathbf{c}_j) = j$, for all $j \in \{0,1,2,3\}$, using (10). The bits of a codeword $\mathbf{c}_j$ are $c_{j,i}$, $i \in \{0,1\}$, and $a_{j,i}$ is defined for each $c_{j,i}$ as in (9).

$$g(\mathbf{c}_0) = \sum_{i=0}^{1} 0 \cdot N(i - a_{i+1}x, x) = 0,$$

$$g(\mathbf{c}_1) = \sum_{i=0}^{1} a_i N(i - a_{i+1}x, x) = N(0-0, x) = 1,$$
$$g(\mathbf{c}_2) = \sum_{i=0}^{1} a_i N(i - a_{i+1}x, x) = N(1-0, x) = 2,$$
$$g(\mathbf{c}_3) = \sum_{i=0}^{1} a_i N(i - a_{i+1}x, x)$$
$$= N(1-0, x) + N(0-1, x) = 2 + 1 = 3. \quad (11)$$

Recall that $N(1, x) \triangleq 2$, for all $x \in \{1, 2, \dots\}$.

Assumption: We assume that the following is correct:

$$g(\overline{m}, x, \overline{\mathbf{c}}) = \sum_{i=0}^{\overline{m}-1} \overline{a}_i N(i - \overline{a}_{i+1}x, x), \quad (12)$$

where $\overline{\mathbf{c}} \in \mathcal{AC}_{\overline{m},x}$ and $\overline{m} \in \{1, 2, \dots, m\}$ with the same notation defined before Theorem 2 apply to $\overline{\mathbf{c}}$. The assumption here basically means (10) is correct for all A-LOCO codes $\mathcal{AC}_{\overline{m},x}$, $\overline{m} \in \{1, 2, \dots, m\}$.

To be proved: We prove that:

$$g(m+1, x, \mathbf{c}') = \sum_{i=0}^{m} a'_i N(i - a'_{i+1}x, x), \quad (13)$$

which means we prove that given the base and the assumption, (10) is also correct for the A-LOCO code $\mathcal{AC}_{m+1,x}$.

One more time, we use the group structure introduced in Section II to prove (13). Note that the group structure can be defined for $\mathcal{AC}_{m+1,x}$ as defined for $\mathcal{AC}_{m,x}$. We use the same codeword correspondence in the proof of Theorem 1 for the three groups (with $m+1$ replacing m).

Group 1: The codewords in Group 1 in $\mathcal{AC}_{m+1,x}$ start at index 0, and the corresponding codewords in $\mathcal{AC}_{m,x}$ also start at index 0. Thus, for this group, the shift in codeword indices between a codeword $\mathbf{c}'$ in $\mathcal{AC}_{m+1,x}$ and the corresponding codeword $\mathbf{c}$ in $\mathcal{AC}_{m,x}$ is:

$$g(m+1, x, \mathbf{c}') - g(m, x, \mathbf{c}) = 0. \quad (14)$$

Consequently, and using (12):

$$g(m+1, x, \mathbf{c}') = g(m, x, \mathbf{c}) = \sum_{i=0}^{m-1} a_i N(i - a_{i+1}x, x). \quad (15)$$

Since $\mathbf{c}'$ starts with 0 from the left, $a'_m = 0$. Additionally, $\mathbf{c}'$ and $\mathbf{c}$ share the m RMBs. Thus, (15) can be written as:

$$g(m+1, x, \mathbf{c}') = \sum_{i=0}^{m} a'_i N(i - a'_{i+1}x, x). \quad (16)$$

Group 2: The codewords in Group 2 in $\mathcal{AC}_{m+1,x}$ start right after Groups 1 and 3, and the corresponding codewords in $\mathcal{AC}_{m,x}$ start right after all the codewords that start with 0 from the left. Thus, for this group, the shift in codeword indices between a codeword $\mathbf{c}'$ in $\mathcal{AC}_{m+1,x}$ and the corresponding codeword $\mathbf{c}$ in $\mathcal{AC}_{m,x}$ is:

$$g(m+1, x, \mathbf{c}') - g(m, x, \mathbf{c})$$
$$= N_1(m+1, x) + N_3(m+1, x) - N_1(m, x)$$
$$= N(m, x) + N(m-x-1, x) - N(m-1, x), \quad (17)$$

where the second equality follows from using (4) and (8). Consequently, and using (12):

$$g(m+1, x, \mathbf{c}')$$
$$= N(m, x) + N(m-x-1, x) - N(m-1, x)$$
$$+ \sum_{i=0}^{m-1} a_i N(i - a_{i+1}x, x). \quad (18)$$

Observe that $c_{m-1} = 1$; thus, $a_{m-1} = 1$, while $a_m \triangleq 0$. Using these observations in (18) results in:

$$g(m+1, x, \mathbf{c}')$$
$$= N(m, x) + N(m-x-1, x) - N(m-1, x)$$
$$+ N(m-1, x) + \sum_{i=0}^{m-2} a_i N(i - a_{i+1}x, x). \quad (19)$$

Since $\mathbf{c}'$ starts with 11 from the left, $a'_m = a'_{m-1} = 1$, same as a_{m-1}, while $a'_{m+1} \triangleq 0$. Consequently,

$$N(m, x) + N(m-x-1, x)$$
$$= a'_m N(m - a'_{m+1}x, x) + a'_{m-1} N(m - 1 - a'_m x, x)$$
$$= \sum_{i=m-1}^{m} a'_i N(i - a'_{i+1}x, x). \quad (20)$$

Additionally, $\mathbf{c}'$ and $\mathbf{c}$ share the $m-1$ RMBs. Thus, aided by (20), (19) can be written as:

$$g(m+1, x, \mathbf{c}')$$
$$= \sum_{i=m-1}^{m} a'_i N(i - a'_{i+1}x, x) + \sum_{i=0}^{m-2} a'_i N(i - a'_{i+1}x, x)$$
$$= \sum_{i=0}^{m} a'_i N(i - a'_{i+1}x, x). \quad (21)$$

Group 3: The codewords in Group 3 in $\mathcal{AC}_{m+1,x}$ start right after Group 1, and the corresponding codewords in $\mathcal{AC}_{m-x,x}$ start at index 0. Thus, for this group, the shift in codeword indices between a codeword $\mathbf{c}'$ in $\mathcal{AC}_{m+1,x}$ and the corresponding codeword $\mathbf{c}''$ in $\mathcal{AC}_{m-x,x}$ is:

$$g(m+1, x, \mathbf{c}') - g(m-x, x, \mathbf{c}'')$$
$$= N_1(m+1, x) = N(m, x). \quad (22)$$

Consequently, and using (12):

$$g(m+1, x, \mathbf{c}') = N(m, x) + \sum_{i=0}^{m-x-1} a''_i N(i - a''_{i+1}x, x)$$
$$= N(m, x) + \sum_{i=0}^{m-x-2} a''_i N(i - a''_{i+1}x, x), \quad (23)$$

where the second equality follows from that $\mathbf{c}''$ starts with 0 from the left; thus, $a''_{m-x-1} = 0$. Since $\mathbf{c}'$ starts with $1\mathbf{0}^{x+1}$ from the left, $a'_m = 1$ and $a'_{m-1} = a'_{m-2} = \cdots = a'_{m-x-1} = 0$, while $a_{m+1} \triangleq 0$. Additionally, $\mathbf{c}'$ and $\mathbf{c}''$ share the $m-x-1$ RMBs. Thus, (23) can be written as:

$$g(m+1, x, \mathbf{c}') = \sum_{i=0}^{m} a'_i N(i - a'_{i+1}x, x). \quad (24)$$

From (16), (21), and (24), (13) is proved, which completes the proof by induction for any A-LOCO code $\mathcal{AC}_{m,x}$, $m \geq 2$ and $x \geq 1$. ■

Example 2. *We illustrate Theorem 2 by applying (10) on two different codewords in $\mathcal{AC}_{5,1}$, given in Table I, to compute their indices. The first codeword is* 01111*; thus,* $a_4 = 0$ *and* $a_3 = a_2 = a_1 = a_0 = 1$*. Consequently,*

$$\begin{aligned} g(\mathbf{c}) &= \sum_{i=0}^{4} a_i N(i - a_{i+1}, 1) \\ &= N(3-0,1) + N(2-1,1) + N(1-1,1) + N(0-1,1) \\ &= N(3,1) + N(1,1) + N(0,1) + N(-1,1) \\ &= 7 + 2 + 1 + 1 = 11, \end{aligned}$$

which is indeed the index of this codeword in Table I. The second codeword is 11001*; thus,* $a_4 = a_3 = a_0 = 1$ *and* $a_2 = a_1 = 0$*, while* $a_5 \triangleq 0$*. Consequently,*

$$\begin{aligned} g(\mathbf{c}) &= \sum_{i=0}^{4} a_i N(i - a_{i+1}, 1) \\ &= N(4-0,1) + N(3-1,1) + N(0-0,1) \\ &= N(4,1) + N(2,1) + N(0,1) \\ &= 12 + 4 + 1 = 17, \end{aligned}$$

which is indeed the index of this codeword in Table I.

Theorem 2 gives the encoding-decoding rule of A-LOCO codes. In particular, this theorem provides a simple mapping-demapping (both are one-to-one) from the index to the codeword and vice-versa. This simple mapping-demapping represented by (10) is what enables low-complexity encoding and decoding algorithms for A-LOCO codes as we shall see later. The main advantage offered by A-LOCO codes is that they are capacity-achieving asymmetric constrained codes with simple encoding-decoding.

IV. Bridging, Clocking, and Achievable Rates

In this section, we discuss the bridging patterns and the self-clocking of A-LOCO codes. Then, we introduce the rates of A-LOCO codes in the finite-length regime and show that they are capacity-achieving codes.

In fixed-length constrained codes [12], given any two consecutive codewords, bridging patterns are needed to prevent forbidden patterns from appearing on the transition from the first codeword to the following codeword [14]. For example, consider the A-LOCO code $\mathcal{AC}_{5,1}$ given in Table I, if the codewords having indices 13 and 8 are to be written consecutively without bridging, we get the following substream of bits $1000\mathit{101}001$ in which, the forbidden pattern 101 does appear (the pattern is shown in italic).

For symmetric LOCO codes, it was shown in [14] that upon deciding the bridging method, there is a compromise between the maximum protection of the bits at the codeword transitions and the minimum number of bits/symbols to be used for bridging. The optimal bridging method for a symmetric LOCO code with parameter x in terms of bits protection was shown to require $2x$ bridging bits, which results in significant rate loss [14]. Thus, we adopted a suboptimal bridging method for symmetric LOCO codes that is bridging by x no writing, or no transmission, symbols.

For an A-LOCO code with parameter x, only patterns of the form $\{101, 1001, \ldots, 1\mathbf{0}^x1\}$ are forbidden. Thus, bridging with x 0's, i.e., with the pattern $\mathbf{0}^x$, ensures that forbidden patterns do not appear on the codeword transitions except for the case when the RMB of a codeword and the LMB of the next codeword to be written are both 1's. In this case, bridging with x 1's, i.e., with the pattern $\mathbf{1}^x$, is used instead. In summary, our bridging method for A-LOCO codes is:

1) If the RMB of a codeword and the LMB of the next codeword to be written are both 1's, bridge with $\mathbf{1}^x$.
2) Otherwise, bridge with $\mathbf{0}^x$.

It is important here to highlight that the proposed bridging methods is one of the advantages of A-LOCO codes over other constrained codes. In particular, the bridging method is not only optimal in terms of bits protection, but also requires the minimum number of added bits for bridging, which is only x bits for an A-LOCO code $\mathcal{AC}_{m,x}$. Observe that this bridging method is also easy to implement.

Next, we discuss the self-clocking of A-LOCO codes. This feature is required in order to have clock recovery and system calibration [2], [14]. A constrained code is said to be self-clocked if any stream of codewords to be written contains a sufficient number of appropriately-separated transitions after bridging and signaling are applied. Since NRZ signaling is adopted for A-LOCO codes, these transitions are the $0-1$ and $1-0$ transitions in A-LOCO codewords. To achieve self-clocking, we just need to remove the two codewords $\mathbf{0}^m$ and $\mathbf{1}^m$ from an A-LOCO code $\mathcal{AC}_{m,x}$ as this guarantees at least one transition in each A-LOCO codeword.

Definition 2. *A self-clocked A-LOCO code (CA-LOCO code) $\mathcal{AC}^{\mathrm{c}}_{m,x}$, $m \geq 2$, is the A-LOCO code $\mathcal{AC}_{m,x}$ after removing the all* 0*'s and the all* 1*'s codewords. Mathematically,*

$$\mathcal{AC}^{\mathrm{c}}_{m,x} \triangleq \mathcal{AC}_{m,x} \setminus \{\mathbf{0}^m, \mathbf{1}^m\}. \tag{25}$$

Thus, the cardinality of $\mathcal{AC}^{\mathrm{c}}_{m,x}$ is:

$$N^{\mathrm{c}}(m,x) = N(m,x) - 2. \tag{26}$$

We define $k^{\mathrm{c}}_{\mathrm{eff}}$ as the maximum number of consecutive cells between two consecutive transitions (all programmed or all unprogrammed) after a stream of CA-LOCO codewords separated by bridging patterns is written; one bit per cell. Thus, $k^{\mathrm{c}}_{\mathrm{eff}}$ is the length of the longest run of consecutive 1's or 0's in a stream of CA-LOCO codewords separated by bridging patterns. The scenarios under which $k^{\mathrm{c}}_{\mathrm{eff}}$ is achieved are:

$$1\mathbf{0}^{m-1} - \mathbf{0}^x - \mathbf{0}^{m-1}1 \quad \text{and} \quad 0\mathbf{1}^{m-1} - \mathbf{1}^x - \mathbf{1}^{m-1}0.$$

Consequently, $k^{\mathrm{c}}_{\mathrm{eff}}$ is given by:

$$k^{\mathrm{c}}_{\mathrm{eff}} = 2(m-1) + x, \tag{27}$$

which is the same equation satisfied by LOCO codes [14].

Given the cardinality of a CA-LOCO code $\mathcal{AC}^{\mathrm{c}}_{m,x}$, the size of the messages $\mathcal{AC}^{\mathrm{c}}_{m,x}$ encodes is:

$$s^{\mathrm{c}} = \lfloor \log_2 N^{\mathrm{c}}(m,x) \rfloor = \lfloor \log_2 (N(m,x) - 2) \rfloor. \tag{28}$$

TABLE II
THE CODEWORDS OF THE CA-LOCO CODE $\mathcal{AC}^{\mathrm{c}}_{5,1}$ AND THE CORRESPONDING MESSAGES.

Message $\mathbf{b}$	Index $g(\mathbf{c})$	Codeword $\mathbf{c}$
1	0000	00001
2	0001	00010
3	0010	00011
4	0011	00100
5	0100	00110
6	0101	00111
7	0110	01000
8	0111	01001
9	1000	01100
10	1001	01110
11	1010	01111
12	1011	10000
13	1100	10001
14	1101	10010
15	1110	10011
16	1111	11000

Consequently, the rate of a CA-LOCO code $\mathcal{AC}^{\mathrm{c}}_{m,x}$, where x bits are used for bridging as illustrated above, is given by:

$$R^{\mathrm{c}}_{\text{A-LOCO}} = \frac{s^{\mathrm{c}}}{m+x} = \frac{\lfloor \log_2(N(m,x)-2) \rfloor}{m+x}. \tag{29}$$

Observe the following:

1) A CA-LOCO code $\mathcal{AC}^{\mathrm{c}}_{m,x}$ contains all the codewords satisfying the $\mathcal{A}_x$ constraint except the two codewords in $\{\mathbf{0}^m, \mathbf{1}^m\}$. This follows from Definitions 1 and 2.
2) The number of bits added for bridging is x, which does not grow with the code length m. Thus, as $m \to \infty$, the x in the denominator of (29) can be ignored.

From the above two observations, we conclude that CA-LOCO codes are **capacity-achieving constrained codes**. Shortly, we will show that rates approaching the capacity can be achieved with low complexity encoding-decoding.

Example 3. *Consider again the A-LOCO code $\mathcal{AC}_{5,1}$ in Table I. The CA-LOCO code $\mathcal{AC}^{\mathrm{c}}_{5,1}$ is obtained by removing the codewords $\mathbf{0}^m$ (with index 0) and $\mathbf{1}^m$ (with index 20) from $\mathcal{AC}_{5,1}$. For this CA-LOCO code, we have:*

$$k^{\mathrm{c}}_{\text{eff}} = 2(5-1)+1 = 9.$$

The size of the messages $\mathcal{AC}^{\mathrm{c}}_{5,1}$ encodes is:

$$s^{\mathrm{c}} = \lfloor \log_2(N(5,1)-2) \rfloor = \lfloor \log_2 19 \rfloor = 4,$$

where $N(5,1) = 21$ from Example 1 and Table I. All the 16 codewords of $\mathcal{AC}^{\mathrm{c}}_{5,1}$ that have corresponding messages are shown in Table II. From (29), the rate is:

$$R^{\mathrm{c}}_{\text{A-LOCO}} = \frac{4}{5+1} = 0.6667.$$

Note that this is a relatively low rate because of the small value of the code length m.

Table III lists the rates of multiple CA-LOCO codes $\mathcal{AC}^{\mathrm{c}}_{m,x}$ for different values of m and $x \in \{1,2\}$. For the case of $x = 1$, at length $m = 44$ (resp., 76), the rate is 0.8000 (resp., 0.8052). The capacity of $\mathcal{A}_1$-constrained codes is 0.8114 [9], [10]. Thus, at length 76 (resp., 113) bits, the CA-LOCO code is within just 0.8% (resp., 0.6%) from the capacity. For the case of $x = 2$, at length $m = 28$ (resp., 64), the rate is 0.6667 (resp., 0.6818). The capacity of $\mathcal{A}_2$-constrained codes is 0.6942 from the finite-state transition diagram (FSTD). Thus, at length 64 (resp., 123) bits, the CA-LOCO code is within just 1.8% (resp., 0.9%) from the capacity. Higher rates are achievable with higher lengths.

TABLE III
RATES OF CA-LOCO CODES $\mathcal{AC}^{\mathrm{c}}_{m,x}$ FOR DIFFERENT VALUES OF m AND $x \in \{1,2\}$.

Code parameters	Rate	Adder size
$m = 17$ and $x = 1$	0.7778	14 bits
$m = 44$ and $x = 1$	0.8000	36 bits
$m = 76$ and $x = 1$	0.8052	62 bits
$m = 113$ and $x = 1$	0.8070	92 bits
$m = 357$ and $x = 1$	0.8101	
$m = 18$ and $x = 2$	0.6500	13 bits
$m = 28$ and $x = 2$	0.6667	20 bits
$m = 64$ and $x = 2$	0.6818	45 bits
$m = 123$ and $x = 2$	0.6880	86 bits
$m = 244$ and $x = 2$	0.6911	

Remark 1. *A-LOCO codes do not satisfy the complement rule of symmetric LOCO codes in [14, Lemma 3]. Thus, balancing A-LOCO codes incurs a higher rate penalty. To reduce this penalty, almost-balanced A-LOCO codes, with no strict guarantee on the maximum magnitude of the running disparity, can be designed using the ideas in [14].*

V. ALGORITHMS, COMPLEXITY, AND COMPARISONS

In this section, we introduce practical encoding and decoding algorithms of A-LOCO codes. We then discuss the complexity of these algorithms, and make comparisons with other asymmetric and symmetric constrained codes that mitigate ICI in Flash systems.

In coding techniques based on indexing points (here representing codewords), devising simple algorithms to perform the mapping-demapping between the index and the associated point is critical to avoid look-up tables; and thus, to make the technique practical for large sizes. For example, motivated by this observation, the authors of [16] developed simple algorithms to index the points of multi-dimensional constellations. The simple, practical algorithms we introduce in this section are also motivated by the same observation.

Algorithm 1 is the encoding algorithm. While generating a specific codeword $\mathbf{c}$ in the algorithm, we define the RMB of the previous codeword as ζ_0.

Example 4. *We apply Algorithm 1 to encode the message* 1010 *using the CA-LOCO code $\mathcal{AC}^{\mathrm{c}}_{5,1}$ ($m = 5$ and $x = 1$). Recall that $N(1,1) \triangleq 2$, $N(2,1) = 4$, $N(3,1) = 7$, $N(4,1) = 12$, and $N(5,1) = 21$ (see Example 1). From Step 6, $g(\mathbf{c}) = \text{decimal}(1010) + 1 = 11$, which is the initial value of* residual. *The bits of the codeword $\mathbf{c}$ are generated as follows:*

1) *For $i = 4$, and since c_5 is set to 0,* subt_index $= i = 4$ *from Step 10. Now,* residual $= 11 < N(4,1) = 12$. *Thus, c_4 is encoded to 0 from Step 15. Then, the **if** condition in Step 20 is satisfied, and because $c_4 \neq 1$, we bridge with $\mathbf{0}^x$ before c_4 assuming that this is not the first codeword.*
2) *For $i = 3$, and since $c_4 = 0$,* subt_index $= i = 3$ *from Step 10. Now,* residual $= 11 > N(3,1) = 7$. *Thus, c_3*

is encoded to 1 *from Step 17, and* residual *becomes* $11 - 7 = 4$ *from Step 18.*

3) *For* $i = 2$*, and since* $c_3 = 1$*,* subt_index $= i - x = 1$ *from Step 12. Now,* residual $= 4 > N(1,1) \triangleq 2$*. Thus,* c_2 *is encoded to* 1 *from Step 17, and* residual *becomes* $4 - 2 = 2$ *from Step 18.*
4) *For* $i = 1$*, and since* $c_2 = 1$*,* subt_index $= i - x = 0$ *from Step 12. Now,* residual $= 2 > N(0,1) \triangleq 1$*. Thus,* c_1 *is encoded to* 1 *from Step 17, and* residual *becomes* $2 - 1 = 1$ *from Step 18.*
5) *For* $i = 0$*, and since* $c_1 = 1$*,* subt_index $= i - x = -1$ *from Step 12. Now,* residual $= 1 = N(-1,1) \triangleq 1$*. Thus,* c_0 *is encoded to* 1 *from Step 17, and* residual *becomes* $1 - 1 = 0$ *from Step 18.*

As a result of this procedure, the message 1010 *is encoded using the CA-LOCO code* $\mathcal{AC}^{\mathrm{c}}_{5,1}$ *to the codeword* 01111*, which is consistent with Table II.*

Algorithm 1 Encoding CA-LOCO Codes

Input: Incoming stream of binary messages.
Decide the value of x based on system requirements.
Use (2) and (3) to compute $N(i,x)$, $i \in \{1,2,\dots\}$.
Specify m, the smallest i in Step 3 to achieve the desired rate. Then, $s^{\mathrm{c}} = \lfloor \log_2 (N(m,x) - 2) \rfloor$.
for each incoming message $\mathbf{b}$ of length s^{c} **do**
  Compute $g(\mathbf{c}) = \text{decimal}(\mathbf{b}) + 1$. *(binary sequence to decimal integer)*
  Initialize residual with $g(\mathbf{c})$ and c_m with 0.
  for $i \in \{m-1, m-2, \dots, 0\}$ **do** *(in order)*
    if $c_{i+1} = 0$ **then**
      Set subt_index $= i$.
    else
      Set subt_index $= i - x$.
    end if
    if residual $< N(\text{subt\_index}, x)$ **then**
      Encode $c_i = 0$.
    else
      Encode $c_i = 1$.
      residual $\leftarrow$ residual $- N(\text{subt\_index}, x)$.
    end if
    if $i = m - 1$ **then**
      if $\zeta_0 = 1$ and $c_{m-1} = 1$ **then**
        Bridge with x 1's, i.e., $\mathbf{1}^x$, before c_{m-1}.
      else
        Bridge with x 0's, i.e., $\mathbf{0}^x$, before c_{m-1}.
      end if
    end if
  end for
end for
Output: Outgoing stream of binary CA-LOCO codewords. *(to be written on the SLC Flash device)*

Observe that Algorithm 1 has less steps and less computations compared with [14, Algorithm 1] for symmetric LOCO codes. The reason is that all the steps required to avoid the patterns in $\{010, 0110, \dots, 01^x0\}$ are not needed here since these patterns are not forbidden for A-LOCO codes. Thus, the encoding complexity of an A-LOCO code is less than that of the LOCO code with the same m and x.

Algorithm 2 is the decoding algorithm. We refer the reader to Example 2 for more understanding of Algorithm 2.

Algorithm 2 Decoding CA-LOCO Codes

Inputs: Incoming stream of binary CA-LOCO codewords, in addition to m, x, and s^{c}.
Use (2) and (3) to compute $N(i,x)$, $i \in \{1,2,\dots,m\}$.
for each incoming codeword $\mathbf{c}$ of length m **do**
  Initialize $g(\mathbf{c})$ with 0 and c_m with 0.
  for $i \in \{m-1, m-2, \dots, 0\}$ **do** *(in order)*
    if $c_{i+1} = 0$ **then**
      Set add_index $= i$.
    else
      Set add_index $= i - x$.
    end if
    if $c_i = 1$ **then**
      $g(\mathbf{c}) \leftarrow g(\mathbf{c}) + N(\text{add\_index}, x)$.
    end if
  end for
  Compute $\mathbf{b} = \text{binary}(g(\mathbf{c}) - 1)$, which has length s^{c}. *(decimal integer to binary sequence)*
  Ignore the next x bridging bits.
end for
Output: Outgoing stream of binary messages.

On the level of a single message-codeword pair, there exists a single **for** loop on m distinct values for the variable i in both Algorithm 1 (the encoding algorithm) and Algorithm 2 (the decoding algorithm). For each value of i, at most one major arithmetic operation is performed. Thus, the complexity of both algorithms has $O(m)$ on that level. Moreover, the main operations in Algorithm 1 are comparisons/subtractions, while the main operations in Algorithm 2 are additions. The largest result of these operations is the maximum value the index $g(\mathbf{c})$ can take, which is $2^{s^{\mathrm{c}}} - 1$. Consequently, the size of the used adders, which is s^{c}, dictates the complexity of the encoding and decoding algorithms of CA-LOCO codes.

Table III links various finite-length rates of CA-LOCO codes $\mathcal{AC}^{\mathrm{c}}_{m,x}$ for different values of m and $x \in \{1,2\}$ to the associated size of adders required to achieve these rates, which is a crucial complexity measure. For example, for the case of $x = 1$, to achieve a rate ≥ 0.8000 (resp., ≥ 0.8050), adders of size 36 bits (resp., 62 bits) suffice. Moreover, for the case of $x = 2$, to achieve a rate ≥ 0.6667 (resp., ≥ 0.6800), adders of size 20 bits (resp., 45 bits) suffice. Note that the two cases of $\mathcal{AC}^{\mathrm{c}}_{357,1}$ and $\mathcal{AC}^{\mathrm{c}}_{244,2}$ are given in the table only to show how close to capacity CA-LOCO codes can get; that is why the adder size is skipped for both.

Next, we compare A-LOCO codes with other constrained codes used for the same purpose. First, we compare with constrained codes based on finite-state machines (FSMs) and sliding window decoders. FSM-based constrained codes are designed by developing an FSTD that represents infinite sequences satisfying the required constraint. Then, multi-

ple steps are performed to generate the encoding-decoding FSM from the FSTD [2], [10]. To construct FSM-based constrained codes with capacity-approaching rates, typically the FSM gets quite complicated, and so are the encoding and decoding procedures.

As a result, we compare with FSM-based $\mathcal{A}_x$-constrained codes that are known to be practical in terms of complexity. A practical FSM-based $\mathcal{A}_1 = \{101\}$-constrained code has a rate of 0.8000 [10], while practical CA-LOCO codes that are $\mathcal{A}_1$-constrained achieve rates ≥ 0.8050 at moderate lengths. Additionally, a practical FSM-based $\mathcal{A}_2 = \{101, 1001\}$-constrained code has a rate of 0.6667, while practical CA-LOCO codes that are $\mathcal{A}_2$-constrained achieve rates ≥ 0.6800 at moderate lengths. The gain in rate achieved by low-complexity CA-LOCO codes (with adder sizes ≤ 64 bits) compared with practical FSM-based constrained codes reaches 3%, which is a significant rate increase for high rates. Techniques raising the rate with similar or less amounts are highly appreciated in the literature, e.g., raising the rate of FSM-based constrained codes forbidding the patterns in $\{0101, 11101\}$ from $\frac{5}{6}$ to $\frac{6}{7}$, which gives a 2.85% gain [10].

A-LOCO codes also have other advantages over FSM-based constrained codes designed for the same purpose. A-LOCO codes are fixed-length codes. Thus, they do not allow errors to propagate from a codeword into another. Additionally, they also enable parallel encoding and decoding in applications where runtime operations speed is critical.

Second, we compare with the binary asymmetric constrained codes in [9]. In [9], the encoding and decoding are based on the unrank and rank procedures described in [9, Algorithm 1] and [9, Algorithm 2]. While these codes are also enumerative, and thus, can achieve high rates, their encoding and decoding algorithms are more complex than those of A-LOCO codes, which are based on a simple rule described in Theorem 2. Additionally, the codes in [9] only consider the effect of adjacent Flash cells, i.e., can only eliminate the pattern 101 for SLC Flash devices.

Third, we compare with symmetric LOCO codes used for the same goal. In particular, we compare an $\mathcal{A}_x$-constrained code (A-LOCO code) of length m with the $\mathcal{S}_x$-constrained code (LOCO code) of length m. For Flash devices where the goal is only to eliminate the patterns in $\{101, 1001, \ldots, 10^x1\}$, the gain in rate achieved by A-LOCO codes over LOCO codes at the same low complexity and achieving nearly the same performance reaches 16% (resp., 25%) for $x = 1$ (resp., $x = 2$). This gain is expected knowing that the capacity of $\mathcal{S}_1$-constrained (resp., $\mathcal{S}_2$-constrained) codes is 0.6942 (resp., 0.5515) [14]. More details are available in Table III and [14, Table IV]. We note that a similar observation was stated in [10] in the context of MR systems.

Like LOCO codes [14], A-LOCO codes are reconfigurable.

VI. Conclusion

We introduced a new family of asymmetric constrained codes, A-LOCO codes, to improve the performance in Flash memories. Only the detrimental patterns in Flash systems are eliminated in A-LOCO codewords. We derived a recursive formula to compute the cardinality of A-LOCO codes. We presented a simple rule for the mapping-demapping between the lexicographic index and the codeword. This rule allowed practical, low-complexity encoding and decoding algorithms of A-LOCO codes. We illustrated how to optimally bridge and to self-clock A-LOCO codes. We showed that A-LOCO codes are capacity-achieving. The complexity of encoding-decoding A-LOCO codes was studied and comparisons with other constrained codes were presented. These comparisons demonstrated that A-LOCO codes offer a rate-complexity trade-off that is better than other constrained codes used for the same purpose. Non-binary constrained codes for Flash devices with more than two levels and multi-dimensional constrained codes for multi-dimensional storage devices are among the near future research directions. QLC Flash memory evolution is expected to benefit from efficient high rate non-binary asymmetric constrained codes.

Acknowledgment

This research was supported in part by NSF under grant CCF 1717602.

References

[1] D. T. Tang and R. L. Bahl, "Block codes for a class of constrained noiseless channels," *Inf. and Control*, vol. 17, no. 5, pp. 436–461, 1970.

[2] P. Siegel, "Recording codes for digital magnetic storage," *IEEE Trans. Magn.*, vol. 21, no. 5, pp. 1344–1349, Sep. 1985.

[3] B. Vasic and E. Kurtas, *Coding and Signal Processing for Magnetic Recording Systems*. CRC Press, 2005.

[4] K. A. S. Immink, P. H. Siegel, and J. K. Wolf, "Codes for digital recorders," *IEEE Trans. Inf. Theory*, vol. 44, no. 6, pp. 2260–2299, Oct. 1998.

[5] K. A. S. Immink, " Modulation systems for digital audio discs with optical readout," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, Atlanta, Georgia, USA, Mar.–Apr. 1981, pp. 587–589.

[6] M. Qin, E. Yaakobi, and P. H. Siegel, "Constrained codes that mitigate inter-cell interference in read/write cycles for flash memories," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 5, pp. 836–846, Apr. 2014.

[7] S. Kayser and P. H. Siegel, "Constructions for constant-weight ICI-free codes," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Honolulu, HI, USA, Jun.–Jul. 2014, pp. 1431–1435.

[8] V. Taranalli, H. Uchikawa, and P. H. Siegel, "Error analysis and inter-cell interference mitigation in multi-level cell flash memories," in *Proc. IEEE Int. Conf. Commun. (ICC)*, London, UK, Jun. 2015, pp. 271–276.

[9] Y. M. Chee, J. Chrisnata, H. M. Kiah, S. Ling, T. T. Nguyen, and V. K. Vu, "Capacity-achieving codes that mitigate intercell interference and charge leakage in Flash memories," *IEEE Trans. Inf. Theory*, vol. 65, no. 6, pp. 3702–3712, Jun. 2019.

[10] R. Karabed and P. H. Siegel, "Coding for higher-order partial-response channels," in *Proc. SPIE 2605, Coding and Signal Process. for Inf. Storage*, Philadelphia, PA, USA, Dec. 1995, pp. 115–127.

[11] T. Cover, "Enumerative source encoding," *IEEE Trans. Inf. Theory*, vol. 19, no. 1, pp. 73–77, Jan. 1973.

[12] K. A. S. Immink, "A practical method for approaching the channel capacity of constrained channels," *IEEE Trans. Inf. Theory*, vol. 43, no. 5, pp. 1389–1399, Sep. 1997.

[13] V. Braun and K. A. S. Immink, "An enumerative coding technique for DC-free runlength-limited sequences," *IEEE Trans. Commun.*, vol. 48, no. 12, pp. 2024–2031, Dec. 2000.

[14] A. Hareedy and R. Calderbank, "LOCO codes: lexicographically-ordered constrained codes," *IEEE Trans. Inf. Theory*, to be published, doi: 10.1109/TIT.2019.2943244.

[15] A. Hareedy, R. Wu, and L. Dolecek, "A channel-aware combinatorial approach to design high performance spatially-coupled codes for magnetic recording systems," Sep. 2018. [Online]. Available: https://arxiv.org/abs/1804.05504

[16] R. Laroia, N. Farvardin, and S.A. Tretter, "On optimal shaping of multidimensional constellations," *IEEE Trans. Inf. Theory*, vol. 40, no. 4, pp. 1044–1056, Jul. 1994.