**Title**
Novel Coding Strategies for Multi-Level Non-Volatile Memories



**Author**
Sala, Frederic

**Publication Date**
2013

Peer reviewed|Thesis/dissertation

University of California

Los Angeles

# Novel Coding Strategies for Multi-Level Non-Volatile Memories

A thesis submitted in partial satisfaction

of the requirements for the degree

Master of Science in Electrical Engineering

by

**Frederic Sala**

2013

Abstract of the Thesis

# Novel Coding Strategies for Multi-Level Non-Volatile Memories

by

**Frederic Sala**

Master of Science in Electrical Engineering

University of California, Los Angeles, 2013

Professor Lara Dolecek, Chair

Non-volatile memories (NVMs) are the most important modern data storage technology. Despite their significant advantages, NVMs suffer from poor reliability due to issues such as voltage drift over time, overwriting, and inter-cell coupling. This thesis applies coding-theoretic techniques to NVMs in order to improve their reliability and extend their lifetimes. In particular, we focus on two classes of problems: those related to the use of thresholds to read memory cells, and those related to inter-cell coupling in the data representation scheme known as rank modulation.

The first part of the thesis develops the concept of dynamic thresholds. In NVMs, reading stored data is typically done by comparing cell values against a set of predetermined, fixed threshold references. However, due to common NVM problems, fixed threshold usage often results in significant asymmetric errors. To combat these problems, the notion of dynamic thresholds was recently introduced. Such thresholds are allowed to change in order to react to changes in cell value distributions. Thus far, dynamic thresholds have been applied to the reading of binary sequences in memories with single-level cells (SLCs).

In this work, the use of dynamic thresholds for multi-level cell (MLC) memories is explored. A general scheme to compute and apply dynamic thresholds is provided. We derive

a series of performance results, based on both practical considerations and theoretical analysis. We show that the proposed threshold scheme compares favorably with the best-possible threshold scheme. Finally, we develop error-correcting codes that are tailored to take advantage of the properties of dynamic thresholds. Code constructions are provided for different channel models, including those allowing limited and unlimited numbers of errors of varying magnitude limitations.

The second part of this thesis is focused on the application of constrained coding to rank modulation. Rank modulation is an MLC NVM scheme where information is represented by the rankings of charge levels in an entire block of cells, rather than the absolute charge level of any particular cell. This scheme resolves certain NVM problems, including write-asymmetry, as it allows for a transition from any information state to any other solely through the addition of charge to an appropriate subset of cells. However, the scheme still suffers from inter-cell coupling errors. Such errors are due to inadvertent charge level increases in cells whose neighboring cells have significantly larger levels.

We introduce constraints that mitigate the inter-cell coupling problem in rank modulation. These constraints typically limit the differences between the ranks of neighboring elements in a permutation, and thus limit the charge level differences between adjacent cells, reducing inter-cell coupling effects. In particular, we analyze the single neighbor $k$-constraint, where neighboring cells' ranks cannot differ by more than $k$. We provide the best-known bounds for the sizes of sets meeting this constraint, and, for certain cases where the parameter $k$ involves a constant term, we derive exact expressions. We perform an asymptotic analysis. Lastly, we introduce an efficient scheme that allows us to systematically generate constrained permutations.

The thesis of Frederic Sala is approved.

Richard D. Wesel

Lieven Vandenberghe

Lara Dolecek, Committee Chair

University of California, Los Angeles

2013

To Jessie.

# Table of Contents

# CHAPTER 1

# Introduction

This thesis is concerned with applications of coding-theoretic strategies to data storage systems known as non-volatile memories (NVMs). The thesis begins with some brief comments on NVMs, including their importance, implementations, downsides, and common solutions.

## 1.1 Background on Non-Volatile Memories

Recently, NVMs such as Flash, electrically erasable programmable ROM (EEPROM) and phase-change memories (PCM) have become among the most popular and promising data storage technologies. NVMs are commonly used in every type of computing device and, in particular, are ubiquitous in mobile storage. The defining feature of NVMs is their ability to retain data without requiring a power supply. NVMs have many additional advantages over traditional magnetic storage technologies. For example, NVMs experience lower failure rates due to their lack of mechanical parts. They also exhibit very fast seek times, often an order of magnitude faster in comparison to those of older storage technologies [CGOZ99].

NVMs are made up of large arrays of memory cells, where each cell can store one or more bits of information. The earliest NVMs featured cells that were only capable of storing a single bit. These are called single-level cell (SLC) memories, although, in fact, each SLC has two levels. The term SLC refers to the fact that only one level is actively written; the other level is the erased state. In order to maximize storage capacity, efforts have been made to increase the number of bits per cell. Memories with cells representing two bits of information

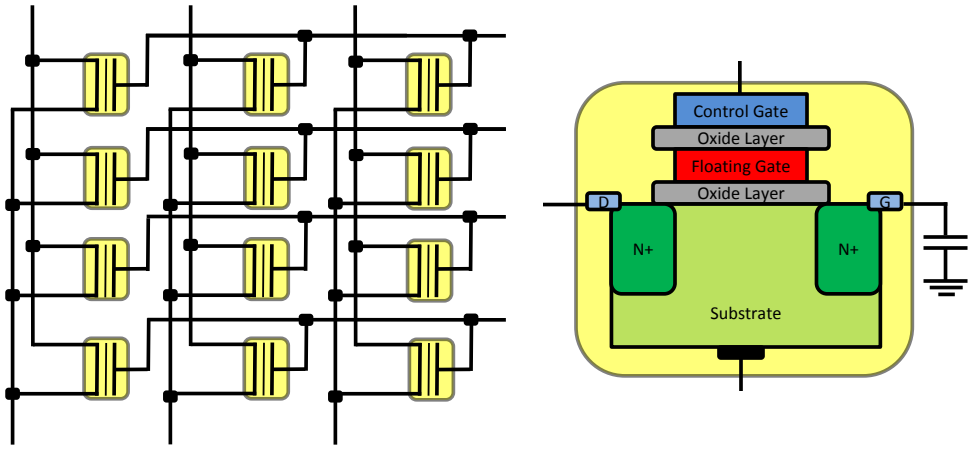


Figure 1.1: Flash cell array schematic and Flash cell block diagram. Cells are floating gate transistors organized in a large rectangular array.

are called multi-level cell (MLC) memories. The term MLC is also applied to cells that store *at least two* bits of information. In this thesis, we will use the term in the latter sense. The current industry standard for Flash memories are cells that store three bits, known as triple-level cells (TLCs).

NVM chips are organized in the following way: Each chip typically contains multiple planes, which are divided into blocks. Each block is further divided into pages. Design considerations, such as the number of cells in a page or a block, are specific to each device's manufacturer. Flash memories have a typical block size of order of magnitude between $10^5$ and $10^6$ cells [CGOZ99, Chapter 3]. In [YGS+12] and [GCC+09], the following design parameters were described for a TLC Flash chip: each block contains 384 pages and each page stores 8 KB of data. Thus, there are 8192000 memory cells in each block.

The mechanism through which cells store information depends on the particular NVM technology used in a device. As an example, we describe the typical Flash technology. Each cell contains a transistor which includes a floating gate in addition to the control gate. The floating gate is insulated by an oxide layer. When electrons are inserted into this gate, they are trapped there. Memory states are then determined by the amount of charge present on

the floating gate. This quantity is determined by measuring the current flow through the transistor. In SLC devices, the two states are simply determined by whether current flows or does not flow. In Figure 1.1, we show a block diagram for a typical Flash cell, along with the schematic for an array of NAND Flash cells.

### Sources of Error in NVMs

To read the value of a memory cell, its voltage is compared to a set of predetermined, fixed reference thresholds. In SLC memories, there is a single threshold $v$ so that a cell value is read as a 0 if its voltage is determined to be below $v$ and as a 1 otherwise[1]. Unfortunately, this approach is prone to errors due to the physical properties of NVMs. For example, in Flash memories, voltage distributions tend to widen over time, expanding beyond the intervals between thresholds. As a result, the distributions overlap and form a potential source of error. This effect, known as charge leakage, is particularly problematic with dense MLCs, where intervals between thresholds are increasingly small.

There are other potential sources of error in NVMs. For example, inter-cell coupling refers to parasitic capacitances between physically adjacent cells. Due to these capacitances, when a large amount of charge is added to a cell, the neighboring cell charge levels can be inadvertently increased, resulting in error [PR04].

NVMs also exhibit asymmetry in the writing process. It is possible to write to (that is, increase the charge levels in) a single cell by injecting electrons into it. However, in order to delete (remove charge from) a cell, an entire block of cells must be deleted [CGOZ99]. The process of erasing a block is time-consuming and permanently damages the device. The lifetime of a Flash NVM allows for about $10^5$ program/erase cycles [BCMV03].

Write-asymmetry is naturally problematic when the same block is frequently rewritten. In addition, since the amount of charge added to a cell when writing is subject to variability, NVM cells are also prone to overwriting. When this occurs, a cell's charge level has been

[1]Note that in actual SLC implementations, for practical reasons, a 0 is read if the voltage is *above* the threshold, and a 1 is read otherwise.

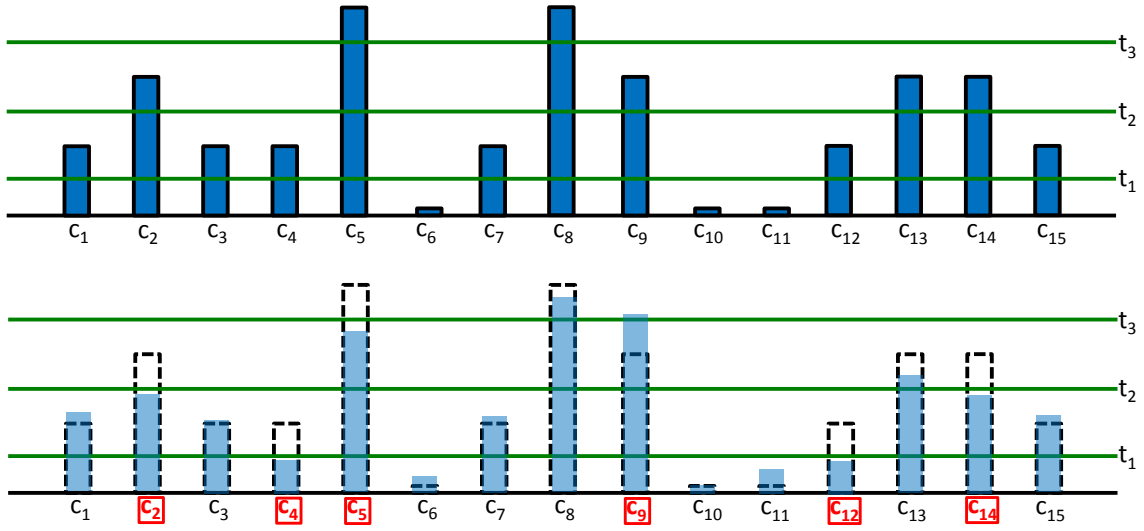


Figure 1.2: Two-bit cell charge levels immediately after writing and after some time. Green lines represent threshold voltages. Boxed cells (highlighted in red) suffer read errors.

brought too high. Removing charge from the cell requires erasing the entire block. For this reason, avoiding overwriting is critical, so charge must be added to cells very gradually, which slows down the writing process.

We illustrate the effects of the previously described sources of error in Figure 1.2. Here, we show a block with 15 MLC cells $c_1, c_2, \ldots, c_{15}$, each storing 2 bits of information. The thresholds $t_1, t_2, t_3$ used when reading are shown as green horizontal lines. The top half of the figure depicts the cells immediately after having been written. In the bottom half, we show the block after a certain duration of time has passed. Now, the cell charge levels, shown in light blue, have changed. The charge levels for cells $c_2, c_4, c_5, c_9, c_{12}$, and $c_{14}$ are no longer in their original threshold intervals. These cells will be read erroneously.

### Coding Strategies

Due to the popularity of NVMs, there have been a large number of research works applying coding-theoretic techniques to NVM storage devices. Most of these coding solutions have focused on resolving the write-asymmetry property of NVM cells. Below, we give a brief

overview of the existing approaches.

*Write-once memory* (WOM) codes were introduced in a seminal paper by Rivest and Shamir [RS82] in 1982. During this early period, WOM codes found use in magnetic and optical media storage systems. These codes are developed for a model where the codeword components can only be increased, never decreased. The ideas in [RS82] were generalized and extended in [FS84]. Some time later, the capacity of generalized WOM codes was determined in [FHV99].

The increase-only criterion used in WOMs is equivalent to the write-asymmetry condition, allowing for the application of WOM codes to NVMs. This has been a particularly fruitful area of research. Efficient multiple-write WOM constructions are given in [KYS$^+$10]. In [Shp12], capacity-achieving binary WOM constructions are developed. In [YS12] and [GDon], bounds and constructions for high-rate non-binary WOM codes are provided.

The *rank modulation* scheme was also developed to alleviate write-asymmetry. In rank modulation, information is not represented by the charge level in a single cell, but rather by the relative rankings of the charge levels of all the cells in a block [JMB09]. These cell ranks induce a permutation, so that rank modulation is a form of permutation coding. The scheme resolves the write-asymmetry issue in the following way: to transition from any permutation to any other permutation, it is only necessary to add charge to an appropriate subset of cells. Overwriting is also resolved, as accidentally increasing cell $a$'s charge level beyond that of cell $b$ can be trivially resolved by adding sufficient charge to cell $b$.

Error-correction coding for permutations has previously been explored in [CK69]. More recently, there have been a series of papers dedicated to studying coding for the rank modulation scheme in particular. Codes correcting a single error are developed in [JSB10]. In [TS10], code constructions which correct limited-magnitude rank modulation errors are generated. In [FSM12], bounds and constructions are provided for codes correcting transposition errors, where the values of two adjacent cells are flipped. Finally, a framework for developing codes over permutations from standard Hamming error-correcting codes is given in [BM10].

With this approach, it is possible to correct any type of error over permutations. A series of constructions with various properties are derived using this technique in [MBZ13].

Note that certain problems affecting NVMs, such as charge leakage and overwriting, result in asymmetric errors. Here the term asymmetric refers to the fact that the errors exclusively increase (or exclusively decrease) cell values. For example, charge leakage always results in a decrease in cell charge levels, while overwriting always increases cell charge levels.

The presence of such asymmetric errors in NVMs has renewed interest in the asymmetric channel, first studied in its simplest form as the Z-channel. Codes for the asymmetric channel have a long history, starting with Berger's error-detection codes [Ber61] in the early 1960s. Recently, a large number of works have focused on the general problem of codes correcting one or more asymmetric errors. In [CSBB10], codes were developed to correct a fixed number of asymmetric (and symmetric) limited-magnitude errors. The limited magnitude error model is particularly appropriate for NVMs, as increases and decreases in charge levels are typically very small. Optimal systematic asymmetric error-correcting code constructions correcting any number of errors were provided in [EB10]. In [KBE11], systematic single asymmetric error-correction constructions were developed. Other works on the subject of asymmetric error-correction include [TB12a], [TB12b], and [Sch12].

Finally, additional NVM code constructions were provided in [MSV+09], [JBB07], and [JLB12]. Application of low-density parity-check (LDPC) codes to Flash memories were explored in [WCSW11] and [ZJB12a]. Solutions focused specifically on phase-change memories were studied in [JZWB11].

The issue of inter-cell coupling, where the difference between cell charge levels may inadvertently result in one of the cells' levels being increased, has also been a focus of coding works. Constrained codes for Flash memories were studied in [BB11]. In this paper, algorithms were developed that allow for writing to cells in a particular order, avoiding writing adjacent cells with significant charge differences consecutively. Applying the notion of constrained coding to the rank modulation scheme described above in order to resolve the

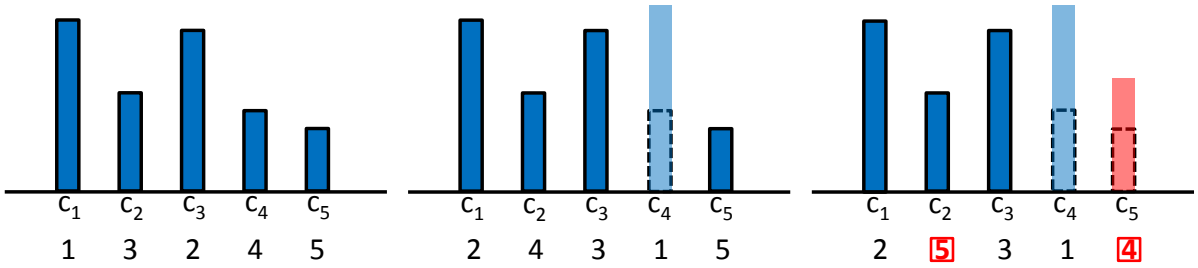


Figure 1.3: Inter-cell coupling in rank modulation. To transition from (1 3 2 4 5) to (2 4 3 1 5), $c_4$'s level must rise. Neighboring cell $c_5$'s level also inadvertently rises, resulting in the incorrect permutation (2 5 3 1 4).

inter-cell coupling issue will be one of the major focuses of this thesis.

An example of a permutation error caused by inter-cell coupling in rank modulation is shown in Figure 1.3. On the left, we show a block with 5 cells $c_1, \ldots, c_5$. The order of the magnitudes of the cells induces the permutation (1 3 2 4 5). Now, if we wish to transition our block to the permutation (2 4 3 1 5), we need only increase the charge level in cell $c_4$ beyond that of cell $c_1$, which is currently the largest level. This is depicted in the central panel of the figure. However, increasing the charge in $c_4$ to such a large extent may cause neighboring cell $c_5$'s charge level to rise. In our example, it has risen beyond the level of $c_2$, as seen on the right side of the figure. Now there is an error: the two permutation elements corresponding to $c_2$ and $c_5$ are incorrect.

An entirely different approach to combating NVM problems (particularly charge leakage) was introduced in [ZJB11]. In this work, Zhou, Jiang, and Bruck introduced the notion of *dynamic thresholds*. With this approach, there are no predetermined thresholds used for comparison when reading cells. Instead, a new set of thresholds is generated each time a cell is read, based on specific information known to the cell decoder (that is, the reader). This technique was shown to be highly effective against errors caused by voltage drift. In this work, we will apply dynamic thresholds to MLC memories and further develop and extend

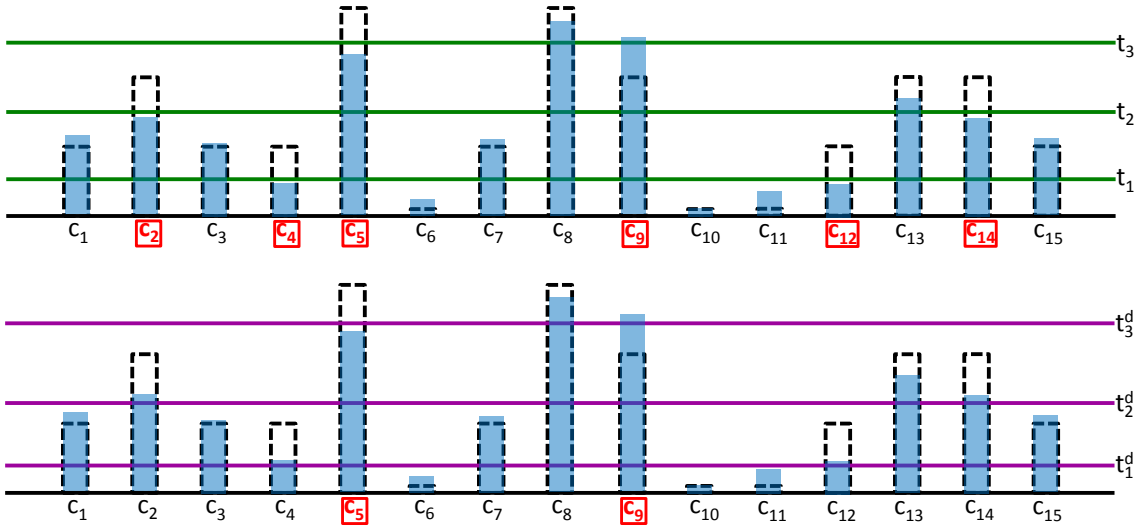


Figure 1.4: Cell charge levels. Top half shows the use of fixed thresholds (in green), causing six errors. Bottom half depicts dynamic thresholds (purple), which have been set according to the block's cell level distributions. Here, only two cells are in error.

this scheme.

Specifically, dynamic thresholds are selected in such a way that the number of cells at each level in a block remains the same when the cells are read as it was when the cells were written. (Of course, this information must be provided to the cell decoder.) We show an example of how this works in Figure 1.4. In the top half, we display the set of 15 cells $c_1, \ldots, c_{15}$ from Figure 1.2. The fixed thresholds, shown in green, cause six cell errors.

In our block, there were originally 2 cells with value 3, 4 cells with value 2, 6 cells with value 1, and 3 cells with value 0. We generate dynamic thresholds so that when we read the cell voltages, we will still have two 3s, four 2s, six 1s, and three 0s overall. Such a set of dynamic thresholds is shown in purple in the bottom half of Figure 1.3. Note that these thresholds are in different positions than the original fixed thresholds. Reading with these thresholds, only two cells, $c_5$ and $c_9$, remain in error, a great improvement in the error rate.

## 1.2 Outline of Contributions

Below, we present a brief outline containing the contributions of this thesis. The first part of the work is centered on the previously-described concept of dynamic thresholds. The second part is concerned with constrained coding in the rank modulation scheme. Our contributions and future directions for our work are summarized in Chapter 4.

### Chapter 2 Contributions

The dynamic thresholding results in [ZJB11] were derived for binary sequences used in the single-level cell (SLC) case. We apply the notion of dynamic thresholds to non-binary sequences, making the technique applicable to MLC memories. We demonstrate that the advantages of dynamic thresholding are preserved in the MLC case. That is, we show that dynamic thresholds allow for lower error rates in comparison to traditional fixed thresholds. We also prove that dynamic thresholds perform close to the "best-possible" threshold scheme. This optimal scheme is not realizable, but serves as a basis for comparison.

Algorithms are introduced to compute dynamic thresholds in MLC NVMs. These algorithms have two flavors. The first is based on the concept of communicating a set of metadata regarding the distribution of levels in a block of cells. The second requires that the block's level distributions are fixed in advance. We also comment on the implementation of dynamic thresholds in real-life devices which suffer from certain limitations.

Finally, we explore the more general problem of combining dynamic thresholding with error-correcting codes in order to guarantee the correction of common errors. Though it is possible to use existing, off-the-shelf codes with the dynamic thresholding scheme, we seek codes that are tailored to correct only those errors which are found when using dynamic thresholds. We study codes that correct a certain number of errors of limited magnitude for each of the computation strategies described above. We also develop codes capable of correcting an unlimited number of limited-magnitude errors.

### Chapter 3 Contributions

In Chapter 3, we introduce the concept of constrained rank modulation. The original rank modulation scheme resolves numerous problems occurring in MLC NVMs. However, it is still prone to error due to the parasitic inter-cell coupling issue. We develop a constrained coding scheme for permutations that resolves this coupling problem in the following way: We introduce constraints that limit the differences between the ranks of adjacent elements in a permutation. As a result, the differences in the charge levels of neighboring cells are also limited. Several constraints with varying strength are introduced.

In particular, we study the single neighbor $k$-constraint, where adjacent elements in the rank modulation permutation may differ by at most $k$. We develop general bounds improving on those in [AK08]. We give an expression for the capacity of codes meeting the constraint. We introduce a code construction which allows us to efficiently generate $k$-constrained permutations. Furthermore, the scheme is shown to be asymptotically optimal.

Finally, we study the sizes of single neighbor constrained codes in two special cases, where the constraint value $k$ is either a constant, or within a constant of the permutation length $n$. We compare the behavior of constrained rank modulation codes in these two extreme cases, which are particularly useful for application to NVM devices.

# CHAPTER 2

# Dynamic Thresholds

## 2.1 Introduction

In this chapter, we study the notion of dynamic thresholds. We open with some preliminaries, including mathematical models and a formal definition of dynamic thresholds. Afterwards, the performance characteristics of the scheme are examined. In order to explore the practical performance aspect, we model cell distributions as Gaussians with certain parameters and derive expressions of error probabilities in some small cases. We also give simulation results using this Gaussian voltage distribution model.

We then compare the particular choice of dynamic thresholds with the "optimal" threshold scheme, which is not realizable, but serves as a good basis for comparison. Finally, we combine dynamic thresholds with error-correcting codes. We seek codes that are specifically designed to correct those errors which occur when using the dynamic thresholding scheme.

We begin by introducing some necessary notation. We will be examining blocks of cells in non-volatile memories. We assume that each cell can take on $q$ levels from the set $F = \{0, 1, 2, \ldots, q-1\}$. Let $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in F^n$ be the word written into a block of cells of length $n$. We let $\mathbf{v}(\mathbf{x}) = (v_1, v_2, \ldots, v_n)$ represent cell levels after writing $\mathbf{x}$, where $v_i \in \mathbb{R}$ for $i \leq i \leq n$. The $v_i$'s tend to vary as a function of time due to the various physical effects experienced by devices discussed earlier.

We are particularly interested in reading a block of cells. To do so, we must interpret the

block's cell voltage vector $\mathbf{v}(\mathbf{x})$ according to a set of reference voltages called a threshold. We define the notion of a threshold vector:

**Definition 1.** *A threshold vector* $\mathbf{t} = (t_1, t_2, \ldots, t_{q-1})$ *is a vector used when reading a block of cells in the following way: Read the word* $\mathbf{y}(\mathbf{t}, \mathbf{v}(\mathbf{x})) = (y_1, y_2, \ldots, y_n)$ *such that* $y_i = m$ *if* $t_m \leq v_i < t_{m+1}$*, for* $0 \leq m \leq q-1$ *and* $1 \leq i \leq n$*, where* $t_0 = -\infty$ *and* $t_q = +\infty$*.*

When it is clear which block $\mathbf{x}$ and corresponding voltage vector $\mathbf{v}(\mathbf{x})$ we are referring to from the context, we remove them from the expression for the output vector $\mathbf{y}$ and write $\mathbf{y}(\mathbf{t})$ or simply $\mathbf{y}$.

We seek to minimize the number of errors that occur when reading the original values in a block $\mathbf{x}$ as $\mathbf{y}$. We will be using the Hamming distance $N(\mathbf{x}, \mathbf{y})$ between $\mathbf{x}$ and $\mathbf{y}$. In words, $N(\mathbf{x}, \mathbf{y})$ is the number of positions $i$, $1 \leq i \leq n$, such that $y_i \neq x_i$. For example, if $\mathbf{x} = (1, 0, 2, 2)$ and $\mathbf{y} = (1, 1, 2, 0)$, $N(\mathbf{x}, \mathbf{y}) = 2$.

We often compare the performance of different thresholds $\mathbf{t}$ for a particular block $\mathbf{x}$ and corresponding $\mathbf{y}$. When we do so, we drop the $\mathbf{x}$ and $\mathbf{y}$ from the expression for distance and write $N(\mathbf{x}, \mathbf{y}) = N(\mathbf{x},\ \mathbf{y}(\mathbf{t}, \mathbf{v}(\mathbf{x})))$ as $N(\mathbf{t})$. Note that $N(\mathbf{x}, \mathbf{y})$ is not the only possible measure of distance between (typically) long vectors $\mathbf{x}, \mathbf{y}$. For example, the $L_1$ distance is defined as $L_1(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} |x_i - y_i|$. Similarly, we can define the asymmetric distance $d_a(\mathbf{x}, \mathbf{y}) = \max(A(\mathbf{x}, \mathbf{y}), A(\mathbf{y}, \mathbf{x}))$, where $A(\mathbf{x}, \mathbf{y}) = |\{i \mid x_i > y_i\}|$.

Next, if we read $\mathbf{x}$ as $\mathbf{y}$ and there exists an entry $i$ with $1 \leq i \leq n$ such that $x_i = a$ and $y_i = b$ with $a \neq b$, we say that an $a \to b$ error has occurred. The number of entries $i$ where $a \to b$ errors occur is denoted $N^{a,b}(\mathbf{t})$, so that $\sum_{a \neq b} N^{a,b}(\mathbf{t}) = N(\mathbf{t})$.

Define $\mathbf{k}(\mathbf{x}) = (k_0, k_1, \ldots, k_{q-1})$ to be a vector such that $k_a$ of the components of $\mathbf{x}$ are at level $a$ for $0 \leq a \leq q-1$. It is clear that $\sum_{a=0}^{q-1} k_a = n$. We are now ready to introduce the concept of the dynamic threshold. Dynamic thresholds are a class of threshold vectors that satisfy the following property:

**Definition 2.** *A dynamic threshold* $\mathbf{t}^d$ *is any threshold vector such that*

$$\mathbf{k}(\mathbf{x}) = \mathbf{k}(\mathbf{y}(\mathbf{t}^d)).$$

*That is,* $\mathbf{t}^d$ *is a threshold vector chosen so that the number of components at each level is preserved from the input vector* $\mathbf{x}$ *to the output vector* $\mathbf{y}(\mathbf{t}^d)$*.*

Additionally, we use the notation $[a, b]$ for the set $\{a, a+1, \ldots, b\}$ with $a \leq b$ and $[n]$ for the set $\{1, 2, \ldots, n\}$.

We illustrate these notions with an example. Take $n = 5$, $q = 3$, and $\mathbf{x} = (1, 0, 2, 2, 0)$. After some time, the cell voltages are given by $\mathbf{v}(\mathbf{x}) = (1.6, 0.3, 2.3, 1.7, 0.7)$. Examining $\mathbf{x}$, we see that $\mathbf{k}(\mathbf{x}) = (2, 1, 2)$. That is, there are two 0s, one 1, and two 2s in $\mathbf{x}$.

Consider the threshold vectors $\mathbf{t}_1 = (0.5, 1.5)$ and $\mathbf{t}_2 = (0.8, 1.65)$. Reading $\mathbf{v}$ with $\mathbf{t}_1$ we have $\mathbf{y}_1 = (2, 0, 2, 2, 1)$ and $\mathbf{k}(\mathbf{y}_1) = (1, 1, 3)$. Then $\mathbf{k}(\mathbf{y}_1) \neq \mathbf{k}(\mathbf{x})$ and $\mathbf{t}_1$ is *not* a dynamic threshold. However, $\mathbf{y}_2(\mathbf{t}_2) = (1, 0, 2, 2, 0)$, and $\mathbf{k}(\mathbf{y}_2) = (2, 1, 2) = \mathbf{k}(\mathbf{x})$ so $\mathbf{t}_2$ is a dynamic threshold.

## 2.2 Performance Analysis

In this section, we seek to motivate the use of dynamic thresholds by demonstrating their performance improvement against fixed thresholds.

### 2.2.1 Practical Considerations

In [ZJB11], it was shown that for binary sequences, dynamic thresholds are particularly effective (when compared to fixed thresholds) when the cell levels are modeled by Gaussian distributions and have variance increasing with time. The same result holds in the non-binary case.

A simple example of this scenario is depicted in Figure 2.1. Here, a block contains only two cells $x_1$ and $x_2$ with true values (that is, the values originally written to $x_1$ and $x_2$) $a$

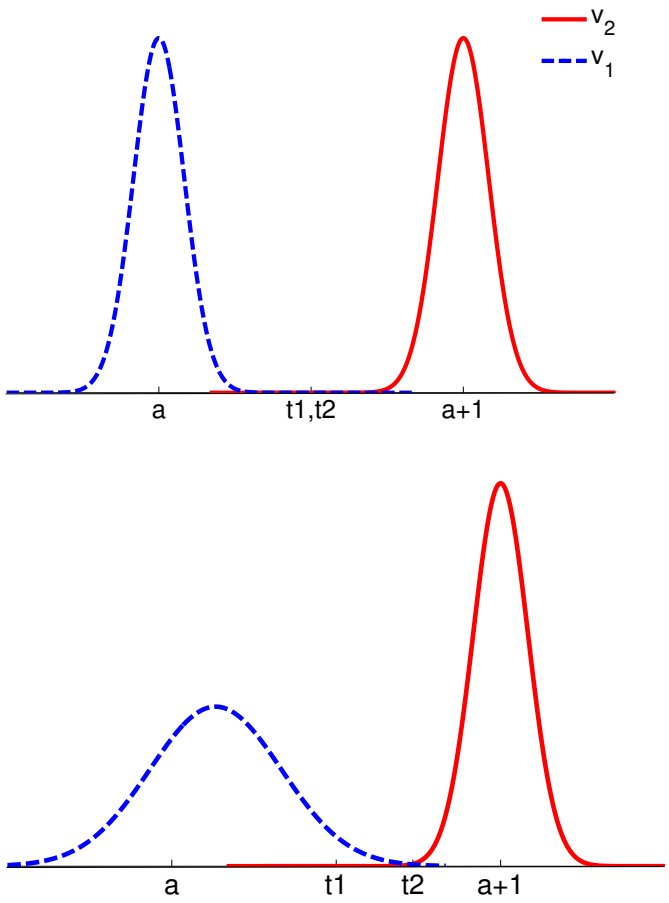


Figure 2.1: Distributions $v_1$ and $v_2$ of cells $x_1$ and $x_2$ with true values $a$ and $a+1$, shown immediately after writing and after some time. Initially, $v_1 \sim \mathcal{N}(a, \sigma^2)$ and $v_2 \sim \mathcal{N}(a+1, \sigma^2)$. After a period of time, $v_1$ has expanded and its mean has shifted. Note the fixed threshold $t_1$ and the dynamic threshold $t_2$: $t_1$ cannot react to the change in $v_1$, but $t_2$ does.

and $a+1$, respectively. The distributions $v_1$ and $v_2$ of these two cells are shown immediately after writing and after some time has passed. The distributions are modeled as independent Gaussians. These Gaussians have the same variance initially, but after a period of time, the variance of $v_1$ increases, and its mean is shifted. The fixed threshold $t_1$ cannot adapt to these changes, leading to a source of error. On the other hand, $t_2$ is shifted to minimize the error probability. This observation provides the fundamental motivation for our use of dynamic thresholding schemes.

We illustrate the strength of dynamic thresholds with the following sample analysis. For

$q$-level cells, we consider the fixed threshold

$$\mathbf{t} = \left(\frac{1}{2}, \frac{3}{2}, \ldots, \frac{2q-3}{2}\right).$$

The voltage written into cell $h$, $v_h$ $(1 \leq h \leq n)$, is read as $a$ if $a - \frac{1}{2} \leq v_h < a + \frac{1}{2}$. For simplicity, in this example we model the $v_h$ as (independent) Gaussians with identical variance: $v_h \sim \mathcal{N}(a, \sigma^2)$, where $a$ is the original value written to $x_h$.

In order to keep our analysis tractable, we take $n = 2$, so that the block $\mathbf{x}$ has only two cells. Let $x_1$ and $x_2$ have values $a$ and $a+1$, respectively, so that $0 < a < q-2$. If we use the threshold $\mathbf{t}$ to read the word, errors occur when the voltage values $v_1$ and $v_2$ fall outside of the $a$th and $(a+1)$st threshold intervals, respectively. That is, errors occur when $v_1 < a - \frac{1}{2}$, $v_1 \geq a + \frac{1}{2}$, $v_2 < (a+1) - \frac{1}{2}$, or $v_2 \geq (a+1) + \frac{1}{2}$. Thus,

$$\Pr\{\text{error}|\mathbf{t}\} = \quad \Pr\left\{v_1 < a - \frac{1}{2} \cup v_1 \geq a + \frac{1}{2} \cup\ v_2 < (a+1) - \frac{1}{2} \cup v_2 \geq (a+1) + \frac{1}{2}\right\},$$

which, after some manipulation, is seen to be equal to

$$2\left(1 - \Phi\left(\frac{1/2}{\sigma}\right) + \Phi\left(\frac{-1/2}{\sigma}\right)\right),$$

where $\Phi(x)$ is the c.d.f. for a zero-mean, unit-variance Gaussian distribution.

Instead, if we employ a dynamic threshold $\mathbf{t}^d$, errors take place when $v_1 \geq v_2$, since if $v_1 < v_2$, the $a$th component of the threshold vector $\mathbf{t}^d$, $t_a$, will be placed between $v_1$ and $v_2$, and $x_1$ and $x_2$ will be read correctly. Then,

$$\begin{aligned}\Pr\left\{\text{error}|\mathbf{t}^d\right\} = & \quad \Pr\{v_i \geq v_j\} \\ = & \quad \int_{-\infty}^{\infty} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(z-(a-1/2))^2}{2\sigma^2}} \Phi\left(\frac{z-(a+1/2)}{\sigma}\right) dz.\end{aligned}$$

This quantity is much smaller than the fixed threshold probability. For example, if $\sigma = 0.25$,

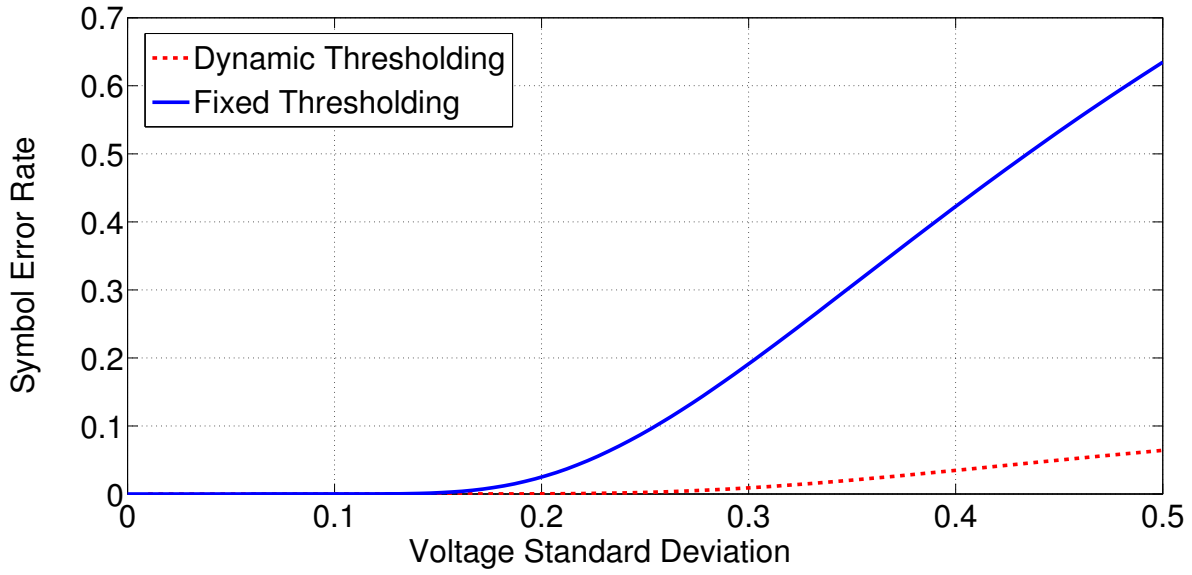


Figure 2.2: Symbol error rate for a pair of cells with adjacent values $a$ and $a+1$ as a function of $\sigma$ for fixed threshold reading versus dynamic threshold reading.

the fixed threshold error probability is $\approx 0.09$ and the dynamic threshold error probability is $\approx 0.0023$. The error probability as a function of the deviation is shown in Figure 2.2.

The previous derivation of error probabilities explains the advantage of dynamic thresholds. When using fixed thresholds, a cell can be in error simply by having its charge level deviate enough to leave its initial threshold interval. With dynamic thresholds, however, cells have to deviate relative to each other to cause an error. That is, two cells' voltage levels must cross over before an error can take place.

Of course, taking the block length to be $n = 2$ is an extreme case. Here, the value of the information $\mathbf{k}(\mathbf{x})$ regarding the two cells' levels is very significant, so that error rates are dramatically improved. Taking the block size to be larger reduces the amount of information provided by this quantity. Nevertheless, the special case $n = 2$ is useful because it provides us with a tractable analysis of the error probability.

We performed simulations to determine whether dynamic thresholds are still effective when the block size is approximately the typical block size in a real device. In Figure 2.3, we provide a plot of the results of the simulation. Here, we used triple-level cells (TLCs), so that $q = 8$, a block length of $n = 10^5$, and voltages modeled as Gaussians with increas-

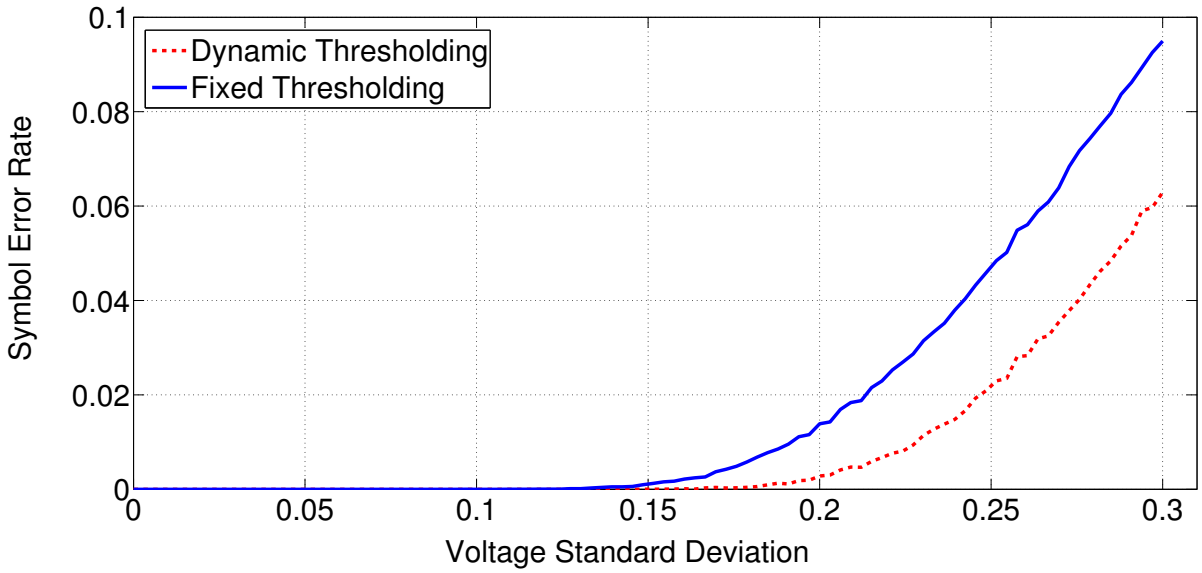


Figure 2.3: Symbol error rate for a block of $10^5$ TLC ($q = 8$) cells using dynamic thresholding and fixed thresholding. Here, voltages are modeled as Gaussians with increasing standard deviation and shifting means.

ing standard deviation and shifted mean over time. The results confirm the advantage of dynamic thresholds versus fixed thresholds in the typical block-length regime.

### 2.2.2 Comparison with the Optimal Scheme

Next, we compare our choice of thresholding scheme with the optimal scheme. Define $\mathbf{t}^* = \arg\min_{\mathbf{t}} N(\mathbf{t})$. We note that computing $\mathbf{t}^*$ requires the knowledge of the initial word $\mathbf{x}$, so the decoder cannot use the optimal threshold. Even so, the dynamic threshold $\mathbf{t}^d$ performs worse than the optimal threshold at most by a fixed factor (which depends on the number of cell levels $q$ and the maximum error magnitude.) In [ZJB11], it was shown that this factor is 2 when $q = 2$, such that $N(\mathbf{t}^d) \leq 2N(\mathbf{t}^*)$. We derive general bounds for multi-level cells.

We define $l$ to be the maximum error magnitude under the dynamic thresholding approach. In general, magnitude limitations model physical limits on the deviation of the voltages $v_i$. However, to keep our analysis tractable, we impose this limitation on the decoded output instead:

**Definition 3.** *Given a block of cells written as* $\mathbf{x}$ *and read as* $\mathbf{y}$ *with a dynamic threshold, we say that errors were of limited magnitude* $l$ *if* $|y_i - x_i| \leq l$ *for* $1 \leq i \leq n$.

This is a reasonable choice: in order for cell $i$ with true value $a$ to experience an error of magnitude $k$ under dynamic thresholding, its voltage $v_i$ must be larger than the voltages of all cells with values $a$, $a+1$, $\ldots$, $a+k-1$ or smaller than the voltages of all cells with values $a, a-1, \ldots, a-k+1$. Thus $k$ must be very small. We begin our analysis with the case $l = 1$, which is of practical interest due to the frequency of such errors in non-volatile memories.

Our approach is to find error patterns that cause a fixed number of errors when using dynamic thresholding and to lower bound the number of errors caused by these patterns when using the optimal threshold. Recall that the number of errors from cell value $a$ to cell value $b$ when reading using threshold $\mathbf{t}$ is denoted by $N^{a,b}(\mathbf{t})$. When $l = 1$, we note that

$$N^{a,b}(\mathbf{t}^d) = N^{b,a}(\mathbf{t}^d).$$

This is easy to establish through an inductive argument: $N^{a,b}(\mathbf{t}^d) = 0$ if $|a-b| > 1$, so it is enough to show that $N^{a,a+1}(\mathbf{t}^d) = N^{a+1,a}(\mathbf{t}^d)$. Take $a = 0$ for the base case. It is clear that (under dynamic thresholding) $N^{0,1}(\mathbf{t}^d) = N^{1,0}(\mathbf{t}^d)$, since the only possible errors involving level 0 are $0 \to 1$ and $1 \to 0$. Assume that $N^{k+1,k}(\mathbf{t}^d) = N^{k,k+1}(\mathbf{t}^d)$. Now, the number of cells at level $k+1$ must not change, so the number of cells transitioning away from $k+1$ must be equal to the number of cells transitioning to $k+1$:

$$N^{k+1,k}(\mathbf{t}^d) + N^{k+1,k+2}(\mathbf{t}^d) = N^{k,k+1}(\mathbf{t}^d) + N^{k+2,k+1}(\mathbf{t}^d).$$

Subtracting the induction hypothesis, we have that

$$N^{k+1,k+2}(\mathbf{t}^d) = N^{k+2,k+1}(\mathbf{t}^d),$$

as desired. This fact leads us to the following theorem:

**Theorem 1.** *For any cell levels $v_1, v_2, \ldots, v_n$, with errors of magnitude limited to $l = 1$, and any dynamic threshold $\mathbf{t}^d$ and optimal threshold $\mathbf{t}^*$,*

$$N(\mathbf{t}^d) \leq 2N(\mathbf{t}^*).$$

*Proof.* From the above result, when using dynamic thresholds, for every error $a \to a+1$, there is a corresponding $a+1 \to a$ error. For such errors, there exists a cell $x_i = a$ with $v_i \geq t_a$ and a cell $x_j = a+1$ with $v_j < t_a$. (Here, $t_a$ refers to the $a$th component of $\mathbf{t}^d$.) Thus, $v_j < v_i$. We have the following (disjoint) possibilities for $t_a^*$ :

$$\begin{aligned} t_a^* &\leq v_j < v_i, \text{ or} \\ v_j &< t_a^* \leq v_i, \text{ or} \\ v_j &< v_i < t_a^*. \end{aligned}$$

The first case will result in an $a \to a+1$ error, the second in both $a \to a+1$ and $a+1 \to a$ errors, and the third in an $a+1 \to a$ error. Thus, for each pair of errors in the scheme using $\mathbf{t}^d$, there is at least one error when using $\mathbf{t}^*$, so that $N(\mathbf{t}^d) \leq 2N(\mathbf{t}^*)$. □

At worst, the use of our particular choice of dynamic thresholds results in twice as many errors as the optimal threshold. We see that (when $l = 1$) the dynamic threshold scheme works as well for $q > 2$ as in the binary case. Notice the following argument: if there exists a pair of cells with levels $a_1, a_2$ with $a_1 < a_2$ and they suffer errors $a_1 \to b_1$ and $a_2 \to b_2$ where $b_1 > b_2$, the optimal threshold scheme will result in at least one error. The case $a_2 = a_1 + 1$ gives the above proof. We exploit this notion to derive a similar bound in the more general case where $1 < l \leq q$.

By definition, dynamic thresholds do not change the distribution of the values of cells in the stored word. The received sequence is then a permutation of the original word, that is,

a multiset permutation. Every multiset permutation can be expressed as a product of cycles [Sta00]. Then, any error $a \to b$ is part of a cycle of $k$ errors

$$a_1 \to a_2 \to a_3 \to \ldots \to a_k \to a_1,$$

where $2 \leq k \leq q$. That is, there exist cells at positions $s_1, s_2, \ldots, s_k$ with levels $a_1, a_2, \ldots, a_k$, respectively, such that cell $x_{s_i}$ suffers an error $a_i \to a_{i+1}$ if $1 \leq i < k$ and cell $x_{s_k}$ suffers an error $a_k \to a_1$.

We illustrate this idea with an example for $n = 3$, $q = 4$, and $l = 2$. Let $(2, 1, 3)$ be written into the cells $(x_1, x_2, x_3)$. After some time, the voltages representing the cells are determined to be $(v_1, v_2, v_3) = (2.4, 1.9, 1.8)$. The decoder knows that there was one cell at level 1, one at level 2, and one at level 3. To preserve the number of cells at these levels, it must pick dynamic thresholds satisfying $1.8 \leq t_1 < 1.9$, $1.9 \leq t_2 < 2.4$, and $2.4 \leq t_3$. Using these thresholds to read the $v_i$'s results in $\mathbf{y} = (3, 2, 1)$, a permutation of the original sequence $(2, 1, 3)$. The errors are given by the cycle $1 \to 2 \to 3 \to 1$.

We are now ready to prove the general bound:

**Theorem 2.** *For any cell levels $v_1, v_2, \ldots, v_n$, with errors of magnitude limited to $l > 1$,*

$$N(\mathbf{t}^d) \leq (l+1)N(\mathbf{t}^*).$$

*Proof.* Any error must be part of a cycle. Consider the cycle

$$a_1 \to a_2 \to a_3 \to \ldots \to a_k \to a_1$$

of length $l < k \leq q$. Let $a_{\min} = \min\{a_1, a_2, \ldots, a_k\}$ and $a_{\max} = \max\{a_1, a_2, \ldots, a_k\}$. Since the $a_i$'s are distinct, $a_{\max} \geq a_{\min} + (k-1)$. Now, split up the interval $[a_{\min}, a_{\max}]$ into $\lceil \frac{k-1}{l} \rceil + 1$ intervals (where each interval, other than the first, has length $l$) in the following

way:

$$[a_{\min}, a_{\min}], [a_{\min}+1, a_{\min}+l], [a_{\min}+l+1, a_{\min}+2l], \ldots, \left[a_{\min} + \left(\left\lfloor \frac{k-2}{l} \right\rfloor\right) l + 1, a_{\max}\right].$$

Note that there are $\lfloor \frac{k-2}{l} \rfloor + 1$ intervals of length $l$, and, including the first interval $[a_{\min}, a_{\min}]$, there are a total of $\lfloor \frac{k-2}{l} \rfloor + 2 = \lceil \frac{k-1}{l} \rceil + 1$ intervals, as desired.

Since there is a path from $a_{\min}$ to $a_{\max}$, there exists $a_i$ in the $f$th interval and $a_j$ in the $(f+1)$st interval such that $a_i \to a_j$. If this was not the case and some interval was skipped, an error would be larger than the interval length of $l$, which is not possible. Similarly, as there is a path from $a_{\max}$ to $a_{\min}$, the cycle contains corresponding descending errors. That is, there exists $a_{i'}$ in the $(f+1)$st interval and $a_{j'}$ in the $f$th interval such that $a_{i'} \to a_{j'}$. Now we use the observation derived in the earlier result. We have that initially there exist cells with values $a_i$ and $a_{i'}$, respectively, such that $a_i < a_{i'}$. These cells are affected by errors of type $a_i \to a_j$ and $a_{i'} \to a_{j'}$ with $a_j > a_{j'}$. The optimal threshold results in one error for each such occurrence, as shown in the proof of Theorem 1.

The cells producing the error above were in the $f$th and $(f+1)$st interval. There is at least one such error for each of the $\left\lceil \frac{k-1}{l} \right\rceil$ pairs of adjacent intervals. Thus, for an error cycle $\mathbf{h}$ of length $k$, usage of the optimal threshold results in at least $\left\lceil \frac{k-1}{l} \right\rceil$ errors. Call the number of errors caused by this cycle $N^{\mathbf{h}}(\mathbf{t})$. Then,

$$N^{\mathbf{h}}(\mathbf{t}^*) \geq \left\lceil \frac{k-1}{l} \right\rceil,$$

while $N^{\mathbf{h}}(\mathbf{t}^d) = k$, so that

$$N^{\mathbf{h}}(\mathbf{t}^d) \leq \frac{k}{\left\lceil \frac{k-1}{l} \right\rceil} N^{\mathbf{h}}(\mathbf{t}^*).$$

Now, for $l \geq 1$, we have that

$$\frac{k}{\left\lceil \frac{k-1}{l} \right\rceil} \leq l + 1,$$

and summing over all error cycles $\mathbf{h}$, we have

$$N(\mathbf{t}^d) = \sum_{\mathbf{h}} N^{\mathbf{h}}(\mathbf{t}^d) \leq \sum_{\mathbf{h}} \frac{k}{\left\lceil \frac{k-1}{l} \right\rceil} N^{\mathbf{h}}(\mathbf{t}^*) \leq (l+1) \sum_{\mathbf{h}} N^{\mathbf{h}}(\mathbf{t}^*) = (l+1) N(\mathbf{t}^*).$$

In the above analysis, we assumed that the cycle $\mathbf{h}$ has length $k > l$. If instead $k \leq l$, there will still be at least two cells whose values are exchanged relative to a threshold, as in our standard argument. Then, as before, using the optimal threshold will result in at least one error. Dynamic thresholds result in $k$ errors for a cycle of length $k$, so

$$N^{\mathbf{h}}(\mathbf{t}^d) \leq k N^{\mathbf{h}}(\mathbf{t}^*) \leq l N^{\mathbf{h}}(\mathbf{t}^*) \leq (l+1) N^{\mathbf{h}}(\mathbf{t}^*),$$

and our result still holds. □

We conclude that, even in the general case, the performance of our scheme is comparable to that of the optimal scheme, which cannot be used without the decoder already knowing the original codeword.

## 2.3 Computing Dynamic Thresholds

Thus far we have analyzed the advantages of replacing fixed thresholds with dynamic thresholds. In this section, we focus on computing dynamic thresholds. First, in order to generate the dynamic thresholds for each codeword, the decoder needs to know the number of cells at each level, given by $\mathbf{k} = (k_0, k_1, \ldots, k_{q-1})$. We give several methods of accomplishing this task. These methods were introduced by Zhou *et al.* in [ZJB11] for the SLC case. We generalize them for MLC memories and suggest some improvements.

### 2.3.1 Metadata Approach

First, we may have the encoder append the base-$q$ representation of each of the $k_i$'s to a transmitted codeword of length $n$. It is sufficient to include $k_i$ for $i < q-1$, as $k_{q-1} = n - \sum_{i=0}^{q-2} k_i$ can be computed from $k_0, k_1, \ldots, k_{q-2}$. Then, as $0 \leq k_i \leq n$, the size of this metadata is $(q-1)\lceil \log_q(n+1) \rceil$. The metadata will be used to generate the dynamic thresholds. For this reason, the metadata must be communicated through a side channel, or, if it is stored in a set of memory cells, it must be decoded using traditional fixed thresholds. Furthermore, since it is crucial that the $k_i$'s are reproduced exactly, the metadata should be protected by a strong error-correcting code.

To compute a dynamic threshold for cell levels $\widetilde{v}_1, \widetilde{v}_2, \ldots, \widetilde{v}_n$, we begin by sorting the $\widetilde{v}_i$'s. Let $v_1 \leq v_2 \leq \ldots \leq v_n$ be the sorted sequence. There must be $k_0$ cells at level 0, so the threshold $t_0$ must be larger than the first $k_0$ values and smaller than the $(k_0+1)$st value. That is, $t_0$ must satisfy $v_{k_0} < t_0 < v_{k_0+1}$. Similarly, there are $k_1$ cells at level 1, so that $t_1$ must be larger than the first $k_0 + k_1$ values, and smaller than the $(k_0 + k_1 + 1)$st smallest value, or $v_{k_0+k_1} < t_1 < v_{k_0+k_1+1}$. Thus, it is natural to select the threshold

$$\mathbf{t}^d = \left( \frac{1}{2}\left(v_{k_0} + v_{k_0+1}\right), \frac{1}{2}\left(v_{k_0+k_1} + v_{k_0+k_1+1}\right), \ldots, \frac{1}{2}\left(v_{k_0+k_1+\ldots+k_{q-2}} + v_{k_0+k_1+\ldots+k_{q-2}+1}\right)\right).$$

This sorting procedure has complexity $O(n \log n)$ in both the average-case and the worst-case. The process does not depend on the number of levels $q$, as the sequence only needs to be sorted a single time.

As seen above, it is in fact only necessary to select the $k_0$th, $(k_0+1)$st, $(k_0+k_1)$st,..., $(k_0+k_1+\ldots+k_{q-2}+1)$st smallest elements from our list, for a total of $2q-2$ selections. There exist algorithms to find the $k$th order statistic with average- and worst-case runtime $O(n)$, such as the BFPRT algorithm in [BFP+73]. Thus, we can find a threshold $\mathbf{t}^d$ with worst-case complexity $O(2qn)$. If $q$ is small compared to $\log n$, this algorithm is more efficient than sorting the list. If $q = 2$, this procedure is better than the proposed algorithm in [ZJB11],

which has worst-case complexity $O(\gamma n)$ for a constant $\gamma$.

### 2.3.2 Balanced Codes Approach

Above, we allowed any possible distribution of levels in our codewords, so that this distribution had to be included as metadata for each transmitted codeword. However, there is another approach. If we *fix* the distribution of levels beforehand, and restrict transmission to codewords having this distribution, we forgo the need for metadata. The decoder will simply be informed beforehand of the chosen distribution. Of course, by fixing this distribution, we lose a number of codeword choices. In order to maximize the size of our codebook, we should pick a "balanced" distribution, where the numbers of cells at each level are as close as possible. For example, if we have $q$ levels and codewords of length $n = eq$, we should have $e$ cells at levels $0, 1, \ldots, q-1$, respectively. This approach is referred to as a balanced code. The number of codewords in the balanced codebook $D_{n,q}$ with codewords of length $n$ over $F = \{0, 1, \ldots, q-1\}$ is

$$|D_{n,q}| = \binom{n}{e, e, \ldots, e} = \frac{n!}{e!e! \ldots e!},$$

where $\binom{n}{e,e,\ldots,e}$ represents a multinomial coefficient. We note that asymptotically, no rate is lost by using the balanced codebook (for fixed $q$), as

$$\lim_{n\to\infty} \frac{\ln |D_{n,q}|}{\ln q^n} = \lim_{n\to\infty} \frac{\ln \frac{n!}{e!e!\ldots e!}}{\ln n!} = 1.$$

Since we need not send metadata, we may save $(q-1)\lceil \log_q(n+1) \rceil$ digits. The downside of this approach is that the codebook is smaller in the finite-length regime and that it is more difficult to construct error-correcting codes over the space of balanced codewords. There is also added complexity in encoding and decoding balanced codes. The binary case of this problem has been studied extensively. It was introduced by Knuth in his seminal paper [Knu86]. Less is known about the non-binary case; generalizations of balanced codes were recently studied in [WISS13], while more general constructions for $q$-ary constant weight

codes were provided in [CL07].

### 2.3.3 Remarks on Real Devices

In this section we comment on the application of dynamic thresholds to real NVM devices, which have certain limitations.

#### Timing of dynamic threshold generation

First, dynamic thresholds have been defined over an entire block of cells. This means that when we wish to read a particular set of cells, we must read the entire block where these cells are found. However, reading an entire block is slow and permanently wears down the device. (Note, however, that wear from reading cells is very small in comparison to wear from erasing cells.)

We solve this problem in the following way: We "refresh" the thresholds in a block using the dynamic thresholds scheme at some time, then use these new thresholds as fixed thresholds until the next time they are refreshed. This allows us to avoid constantly re-reading a block, while still avoiding using inappropriate and outdated fixed thresholds for the entire lifetime of the device.

Of course, it is necessary to determine when the the thresholds should be refreshed. We suggest the following possibilities:

- After a fixed timeout of $\alpha$ seconds
- When an entire block of cells is being read, such as in a copy or search operation. Here, the entire block is read, so there is no delay or wear added by computing thresholds
- After a sufficiently large number of cells are written to
- When the storage system is experiencing little traffic, so that additional delay is not problematic

Simple, practical algorithms exist to check for these conditions. For example, for the third criterion, we need only a single flag and a counter per block of cells. This flag will be set to "clean" and the counter to zero after a dynamic thresholds refresh. The counter is increased each time a cell is written to, and, once a limit is reached, the flag is set to "dirty", triggering a refresh of the thresholds.

The particular parameters used in the previous conditions can be set according to each device's design. For example, in the first criterion, we can decrease error probabilities (at the cost of additional delay and wear) by setting the timeout value to be very small.

### Generating thresholds without discrete voltages

Another issue in current devices is that, in general, cell decoders do not have access to a set of discrete voltage values for each cell. However, in our model, we assume that the decoder knowns these values. As a result, implementation of dynamic thresholds would require a change in the design of NVMs. Instead, we introduce a solution that does not require this potentially complicated change.

Typically, the decoder is simply given the output of a comparison with the threshold levels, so that only the final value in $\{0, 1, \ldots, q-1\}$ is available. Now, we may select *any* set of thresholds $\mathbf{t}$ and compare the resulting level distribution $\mathbf{k}(\mathbf{y}(\mathbf{t}))$ with the desired distribution $\mathbf{k}(\mathbf{x})$. We simply adjust the thresholds accordingly, and repeat this process until $\mathbf{k}(\mathbf{y}(\mathbf{t}))$ is sufficiently close to the distribution $\mathbf{k}(\mathbf{x})$.

For example, if $q = 2$ and we know (according to metadata or to balanced codes) that half of the $n = 100$ cells in a block should be at level 0 and half at level 1, and we use an initial threshold $t^{(0)}$ so that we read 75 1s and 25 0s, we need only use a threshold $t^{(1)}$ which is larger than $t^{(0)}$ by a fixed amount. If we have overshot and now too many cells are 0s, we use $t^{(2)}$ which is between the previous two thresholds, such as, for example, $t^{(2)} = (t^{(0)} + t^{(1)})/2$, and so on. This scheme is similar to a binary search algorithm.

## 2.4 Error-Correction Schemes

Though dynamic thresholds reduce error probabilities, we may also wish to guarantee the correction of common errors. In this section, we develop codes that take advantage of the properties of the dynamic thresholding scheme. When we rely on the metadata implementation approach, for the sake of simplicity, we abstract the metadata by assuming that it has been made available to the decoder with no error.

We note that, in general, we could combine dynamic thresholding with existing, off-the-shelf codes. However, such codes are not specifically tailored to the conditions which occur under dynamic thresholding. For example, errors in dynamic thresholding are limited to those which make the received sequence a multipermutation of the original sequence. We seek codes that fully take advantage of dynamic thresholding conditions.

### 2.4.1 Metadata-based $(t, l)$-DT Error-Correcting Codes

We begin by exploring codes that correct $t$ errors of magnitude limited to $l$, where $1 \leq l \leq q$, $1 \leq t \leq \rho$, and $\rho$ is a constant. We refer to such errors as $(t, l)$-dynamic thresholding errors, or $(t, l)$-DT errors.

We will see that the construction of $(t, l)$-DT error-correcting codes is connected to several known coding-theoretic problems. If our dynamic thresholding implementation uses the metadata approach, the resulting problem is similar to location correction. On the other hand, if we select the balanced codes implementation, the problem resembles coding for rank modulation. In this section, we study the metadata approach.

As described above, we will find that correcting $(t, l)$-DT errors is closely related to the problem of location correction, studied by Roth and Seroussi in [RS96]. In the location correction problem, the decoder is provided with the magnitudes of the errors, but does not know their locations. Our problem differs: the decoder knows the received sequence is a permutation of the original codeword. This fact only provides some information about the

error magnitudes. Despite this, it will be shown the two problems are nearly equivalent.

### Decomposability Distance

Roth and Seroussi introduced the notions of *decomposability* and *decomposability distance* [RS96]. We present a variant of these concepts which incorporates limited magnitudes:

**Definition 4.** *A vector* $\mathbf{x} \in F^n$ *is* $(\tau, l)$*-decomposable if* $x_i$ *has magnitude limited to* $l$[1] *for* $1 \leq i \leq n$ *and there are two vectors* $\mathbf{y}$ *and* $\mathbf{z}$ *in* $F^n$ *with the same multiset of (up to)* $\tau$ *nonzero values such that* $\mathbf{x} = \mathbf{y} - \mathbf{z}$.

Note that here all operations are taken over the ring of integers modulo $q$. Next, we define decomposability distance:

**Definition 5.** *For fixed* $l$*, the decomposability weight of* $\mathbf{x} \in F^n$ *is the smallest non-negative integer* $\tau$ *(if any such* $\tau$ *exists) such that* $\mathbf{x}$ *is* $(\tau, l)-$*decomposable. If no such* $\tau$ *exists, the decomposability weight is defined to be* $n+1$*. The decomposability distance between* $\mathbf{x}$ *and* $\mathbf{y}$ *in* $F^n$ *is defined as the decomposability weight of* $\mathbf{x} - \mathbf{y}$.

Decomposability is useful in the context of location correction, as it naturally describes location error vectors whose component magnitudes are taken from a known, fixed set, while their locations are free to vary. We will see that decomposability distance is also applicable to error vectors in dynamic thresholding. Towards this end, we first give an alternative definition of decomposability.

We define the unit vector $\mathbf{u}_i$ as the vector with a 1 in the $i$th position and zeros elsewhere. If $i \neq j$, we let $\mathbf{u}_{i,j} = \mathbf{u}_i - \mathbf{u}_j$. Denote by $W_F(n)$ the set of all vectors $\mathbf{u}_{i,j} \in F^n$. The following lemma is equivalent to a result in [RS96]:

**Lemma 1.** *A vector* $\mathbf{x} \in F^n$ *is* $(\tau, l)$*-decomposable if and only if its components are limited to magnitude* $l$ *and it can be written as a linear combination of* $\tau$ *elements of* $W_F(n)$*. That*

[1] Although we are working in the set $F = \{0, 1, \ldots, q-1\}$, we will often interpret the set element $a > \frac{q}{2}$ as the negative number $-(q-a)$. In this context, magnitude is defined as $\min\{a, (q-a)\}$. For example, the magnitude of $-1 = q - 1 \in F$ is $|q-1| = |-1| = |1| = 1$.

*is, there exist* $\tau$ *elements* $E_k \in F^n$ *(with* $1 \leq k \leq \tau$*) and respective vectors* $\mathbf{u}_{i_k,j_k} \in W_F(n)$ *such that* $\mathbf{x} = \sum_{k=1}^{\tau} E_k \mathbf{u}_{i_k,j_k}$.

We illustrate these concepts with an example. If we take $\mathbf{x} = (1, 2, 3, 4)$, $\mathbf{y} = (1, 4, 3, 2)$, and $l = 2$, then $\mathbf{x} - \mathbf{y} = (0, -2, 0, 2) = -2\mathbf{u}_{2,4}$, so that $\mathbf{x} - \mathbf{y}$ is $(1, 2)$-decomposable. Thus $\mathbf{x} - \mathbf{y}$ has decomposability weight 1, and $\mathbf{x}$ and $\mathbf{y}$ are at decomposability distance 1. On the other hand, if we had taken $l = 1$, $\mathbf{x} - \mathbf{y}$ is not $(\tau, 1)$-decomposable for any $\tau$, so that the decomposability weight of $\mathbf{x} - \mathbf{y}$ is $n + 1 = 5$, and $\mathbf{x}$ and $\mathbf{y}$ are at decomposability distance $n + 1 = 5$.

Now we are ready to tie together decomposability and dynamic thresholds. In our problem, the received vector $\mathbf{y}$ is a multiset permutation of the input $\mathbf{x}$ with at least $n - t$ fixed points, as there are at most $t$ errors. Consequently, we have the following lemma:

**Lemma 2.** *Under dynamic thresholding, the* $(t, l)$*-DT error vector* $\mathbf{e} = \mathbf{y} - \mathbf{x}$ *is* $(t - 1, l)$*-decomposable.*

*Proof.* Consider the cells at positions $s_1, s_2, \ldots, s_k$ with values $a_1, a_2, \ldots, a_k$, respectively. Say that the values are permuted by the cycle $\mathbf{h}$ of length $k$ such that $a_1 \to a_2 \to \ldots \to a_{k-1} \to a_k \to a_1$. Then, these components in the error vector $\mathbf{e}$ will have values $a_2 - a_1, a_3 - a_2, \ldots, a_k - a_{k-1}, a_1 - a_k$. This contribution can be expressed as

$$\mathbf{h} = (a_2 - a_1)\mathbf{u}_{s_1,s_k} + (a_3 - a_2)\mathbf{u}_{s_2,s_k} + \ldots + (a_k - a_{k-1})\mathbf{u}_{s_{k-1},s_k}.$$

The claim is clearly true for the first $k - 1$ components. The last component (at position $s_k$) is correct, since $-(a_2 - a_1 + a_3 - a_2 + \ldots + a_k - a_{k-1}) = a_1 - a_k$, as desired. Lemma 1 implies that $\mathbf{h}$ is $(k - 1, l)$-decomposable (as, of course, $|a_{i+1} - a_i| \leq l$.)

The error vector $\mathbf{e}$ is the sum of disjoint cycles $\mathbf{h}_i$ of length $k_i$ where $\sum_i k_i = t$. Now, we have that the cycles $\mathbf{h}_i$ are disjoint, each of the $\mathbf{h}_i$'s is $(k_i - 1, l)$-decomposable, and $\sum_i (k_i - 1) \leq t - 1$. Thus, $\mathbf{e}$ is $(t - 1, l)$ decomposable. □

We note some properties of DT error vectors based on this result. First, if the cycles involved in the multiset are all of length 2, that is, transpositions, then the contribution of that cycle to the error vector is a single term of the form $(a_2 - a_1)\mathbf{u}_{s_1,s_2}$, for a transposition of the values $a_1$ and $a_2$ at positions $s_1$ and $s_2$, respectively. This contribution is $(1, l)$-decomposable, and there are at most $t/2$ such transpositions, so an error vector made of transpositions is $(t/2, l)$-decomposable.

It is clear that this is the "tightest" decomposability level - it is not possible for an error vector to be $(e, l)$-decomposable for $e < \frac{t}{2}$. More generally, we see that if the largest cycle in the permutation is of length $k \geq 2$, then the error vector is $(k-1, l)$-decomposable, and not $(e, l)$-decomposable for any $e < k - 1$.

Recall from the proof of Theorem 1 that in the case where $l = 1$ (the error magnitude is limited to 1), the maximum error cycle length is 2. That is, all error vectors are made up of transpositions of adjacent values. Then, in this case, the error vectors are $(\frac{t}{2}, 1)$-decomposable. We rely on this fact in the next section.

### Constructions

Now, consider two codewords $\mathbf{c_1}, \mathbf{c_2}$ and their respective $(t, l)$-DT error vectors $\mathbf{e_1}, \mathbf{e_2}$ under dynamic thresholding. It is possible to produce the same received sequence if $\mathbf{c_1}+\mathbf{e_1} = \mathbf{c_2}+\mathbf{e_2}$, or $\mathbf{c_1} - \mathbf{c_2} = \mathbf{e_2} - \mathbf{e_1}$. Since $\mathbf{e_1}, \mathbf{e_2}$ are $(t-1, l)$–decomposable, their difference is $(2t-2, 2l)$-decomposable, and the decomposability distance between $\mathbf{c_1}$ and $\mathbf{c_2}$ is at most $2t - 2$. We have the following result:

**Lemma 3.** *A code $C$ over $F$ is capable of correcting all $(t, l)$-DT errors if and only if it has minimum decomposability distance $2t - 1$ with respect to $2l$.*

We refer to such a code as a $(t, l)$-dynamic thresholding error-correcting code, or a $(t, l)$-DTEC code.

With the above, we have shown the equivalence of our problem and location correction.

We may now apply Roth and Seroussi's constructions from [RS96] to the dynamic thresholding problem. These constructions were used to build location-correcting codes (LCCs), but they also correct $(t, l)$-DT errors. We begin with a construction for $t = 2$ and $l = q - 1$. This is the smallest positive value of $t$, as a single error is not possible. From Lemma 2, an error vector must be $(1, l)$-decomposable. Recall that a $B_2$ set, also known as a *Sidon* set, is a set $S$ with the property that for any four distinct elements $a, b, c, d \in S$, $a + b \neq c + d$ [BS85]. Then, we have:

*Construction 1*: Let $C$ be a $[n, n-2]$ linear block code (of length $n$ and dimension $n-2$) over $F$ with the parity-check matrix

$$H = \begin{bmatrix} a_1 & a_2 & \dots & a_n \\ a_1^2 & a_2^2 & \dots & a_n^2 \end{bmatrix},$$

where $S = \{a_1, a_2, \dots, a_n\}$ is a subset of distinct elements of $F$. Then, $C$ is a $(2, q-1)$-DTEC code if $S$ is a Sidon set.

*Proof.* If two codewords $\mathbf{c_1} \neq \mathbf{c_2}$ can be confused, $\mathbf{c_1} + \mathbf{e_1} = \mathbf{c_2} + \mathbf{e_2}$, with $\mathbf{e_1} \neq \mathbf{e_2}$. Multiplying by the parity-check matrix $H$, we have $\mathbf{e_1}H^T = \mathbf{e_2}H^T$. Since $\mathbf{e_1}$ and $\mathbf{e_2}$ are $(1, l)$-decomposable, $\mathbf{e_1} = K\mathbf{u}_{b,c}$ and $\mathbf{e_2} = J\mathbf{u}_{d,f}$, with $K, J \in F$. Then,

$$\begin{aligned} K(a_b - a_c) &= J(a_d - a_f), \text{and} \\ K(a_b^2 - a_c^2) &= J(a_d^2 - a_f^2), \end{aligned}$$

and dividing the second equation by the first, we have that $a_b + a_c = a_d + a_f$. But, as $a_b, a_c, a_d, a_f$ belong to the Sidon set $S$, we have reached a contradiction. □

Construction 1 provides a $(2, q-1)$-DTEC code. As we have done before, we examine the common case $l = 1$. Then, we may define a $[n, n-1]$ $(2, 1)$-DTEC code by the parity-check

matrix

$$H = \begin{bmatrix} a_1 & a_2 & \dots & a_n \end{bmatrix},$$

where $S = \{a_1, a_2, \dots, a_n\}$ is a Sidon set. The proof of this claim follows along the same lines as that of Construction 1, but as $l = 1$, we have that $K = J = 1$, and we immediately get the equation $a_b + a_c = a_d + a_f$.

So far, our constructions allow for any error magnitude ($l = q-1$) or an error magnitude of 1. If the error magnitude is limited to $1 < l < q-1$, we can take advantage of this fact by applying a technique introduced by Cassuto *et al.* in [CSBB10]. The proof is very similar to that in [CSBB10]:

*Construction 2*: Let $C_2$ be a $(t, l)$-DTEC code over an alphabet of size $q' = 2l+1$. Then, the code $C_1$ over $F$ defined as

$$C_1 = \{\mathbf{x} \in F^n \mid \mathbf{x} \bmod q' \in C_2\}$$

is a $(t, l)$-DTEC code over an alphabet of size $q > q'$. Here, the notation $\mathbf{x} \bmod q'$ represents $(x_1 \bmod q', x_2 \bmod q', \dots, x_n \bmod q')$.

*Proof.* Due to Lemma 3, it is enough to show that any two codewords $\mathbf{x}, \mathbf{z} \in C_1$ are at decomposability distance of at least $2t-1$. That is, we must show that the decomposability weight of $\mathbf{x} - \mathbf{z}$ (with respect to $2l$) is at least $2t-1$, or, $\mathbf{x} - \mathbf{z}$ is $(k, 2l)$-decomposable only for $k \geq 2t-1$.

If there exists $i \in [n]$ such that $|x_i - z_i| > 2l$, $\mathbf{x} - \mathbf{z}$ has decomposability weight $n+1$, and we are done. If there is no such $i$, $\mathbf{x} - \mathbf{z} = \mathbf{x} - \mathbf{z} \bmod q'$. That is, the differences $|x_i - z_i|$ are identical in the fields of order $q'$ and $q$. As $C_2$ is a $(t, l)$-DTEC code, $\mathbf{x} - \mathbf{z} \bmod q'$ has decomposability weight at least $2t-1$, so $\mathbf{x} - \mathbf{z}$ must also have decomposability weight at least $2t-1$ in the field of order $q$, as desired. □

Taking Construction 1 as the inner code $C_2$ in Construction 2, with alphabet size $2l+1$,

we have a $(2, l)$-DTEC code $C_1$ over an alphabet of size $q$. Let $\mathbf{w}$ be a codeword in $C_2$. Then, if $x_i \equiv w_i \bmod (2l+1)$ for $1 \leq i \leq n$, $\mathbf{x}$ will be a codeword in $C_1$. There are between $\lfloor \frac{q}{2l+1} \rfloor$ and $\lceil \frac{q}{2l+1} \rceil$ choices for $x_i$ for each $\mathbf{w} \in C_2$. Then, we can derive some bounds on the number of codewords $|C_1|$:

$$\left\lfloor \frac{q}{2l+1} \right\rfloor^n |C_2| \leq |C_1| \leq \left\lceil \frac{q}{2l+1} \right\rceil^n |C_2|.$$

For example, if $l = 11$, we have the Sidon set $S = \{0, 1, 2, 4, 7\}$ for a [5,3] inner code $C_2$ of size 8. Then, for $q = 64$, $C_1$ is a $(2, 11)$-DTEC code with codebook size satisfying

$$256 \leq |C_1| \leq 1944.$$

The length of our codes depends on the sizes of Sidon sets. The bound

$$|S|(|S| - 3) + 1 \leq q$$

for $S \subset F = \{0, 1, \ldots, q-1\}$, along with several others are proved in [RS96]. This upper bound is problematic for small $l$ in Construction 2. However, one can take the inner code $C_2$ over a larger field to get a longer code at the expense of additional redundancy.

Next we seek to derive $(t, l)$-DTEC codes for $t > 2$. We do so by relying on generalizations of Sidon sets called $B_t(S)$ sets, studied in [BS85]. In $B_t(S)$ sets, all sums of $t$ arbitrary elements are different. Sidon sets represent the $t = 2$ case. The existence of such sets is shown in a famous number-theoretic result from [BC62]:

**Theorem 3.** *If $s = p^u$ (where $p$ is a prime) and $m = (s^{v+1} - 1)/(s - 1)$, we can find $s + 1$ non-negative integers less than $m$*

$$d_0 = 0, d_1 = 1, d_2, \ldots, d_s,$$

*such that the sums*

$$d_{i_1} + d_{i_2} + \ldots + d_{i_v},$$

*($0 \leq i_1 \leq i_2 \leq \ldots \leq i_v \leq m$), are all different modulo $m$.*

This theorem is frequently cited in the coding literature. It is particularly useful when constructing codes where codewords must meet some *checksum* constraint, such as $\sum_{i=1}^{n} ix_i \equiv a \bmod m$. Such checksum-based codes are capable of correcting "unconventional" errors, such as asymmetric errors, insertion/deletion errors, and repetition errors. They were initially introduced by Varshamov and Tenengolts to correct a single asymmetric error [VT65]. We will use such a construction to develop $l$-DTEC codes based on the balanced codes approach in the next section.

Theorem 3 will enable us to build codes for the case $t > 2$. We give an example for the $l = 1$ case. Recall that the error vectors in this case are $(t/2, 1)$-decomposable, and thus have error value $E = 1$. Then, if we generate a set of $s + 1$ integers $a_0 = 0, a_1, a_2, \ldots, a_s$ using the above theorem, the parity-check matrix

$$H = \begin{bmatrix} a_1 & a_2 & \ldots & a_s \end{bmatrix},$$

defines an $[s, s-1]$ $(t/2, 1)$-DTEC code.

Of course, one can develop general $(t, l)$-DTEC codes with the approach of Construction 2, which does not depend on $t$. Decoders for codes similar to Construction 2 are presented in [CSBB10]. The optimality of codes equivalent to LCCs, such as Construction 1, is shown in [RS96].

### 2.4.2 Balanced $(t, l)$-DT Error-Correcting Codes

We now examine the other implementation approach, based on balanced codes. For the sake of simplicity, we let $n = eq$ for some integer $e$. Then, in a balanced code, the space of possible codewords is

$$D_{n,q} = \{\mathbf{c} \in F^n \mid \mathbf{k}(\mathbf{c}) = (e, e, \ldots, e)\}.$$

That is, the space $D_{n,q}$ is the set of permutations of the multiset

$$\{\underbrace{0,0,\ldots,0}_{e\ 0\text{s}},\underbrace{1,1,\ldots,1}_{e\ 1\text{s}},\ldots,\underbrace{q-1,q-1,\ldots,q-1}_{e\ (q-1)\text{s}}\}.$$

Each codeword is a multiset permutation that contains each symbol $0,1,\ldots,q-1$ exactly $e$ times. Note that the size of $D_{n,q}$ is $\frac{n!}{(e!)^q}$. We seek to construct error-correcting codes over the space $D_{n,q}$. Our approach is inspired by codes over the symmetric group of permutations $S_n$, such as rank modulation codes. In [JSB10], [BM10], and [TS10], error-correcting codes for rank modulation are developed. We will see that a similar approach can be used to construct $(t,l)$-DTEC codes for our balanced codes dynamic thresholding implementation.

We note the fundamental differences between $S_n$ and $D_{n,q}$. $S_n$ is a group under permutation multiplication, defined as function composition. $D_{n,q}$ cannot be defined as a group in this way. Although it is possible to introduce a multiplication operation for elements in $D_{n,q}$, inverses will not be unique. Due to these differences, some care must be taken when generalizing results originally applied to $S_n$.

First, we introduce the Kendall tau distance $d_\tau$ among elements in $D_{n,q}$. This measure of distance is typically applied to elements in $S_n$, but it is equally applicable for codewords in $D_{n,q}$, that is, multiset permutations.

**Definition 6.** *For $\mathbf{c_1},\mathbf{c_2}\in D_{n,q}$, the Kendall tau distance $d_\tau(\mathbf{c_1},\mathbf{c_2})$ is defined as*

$$d_\tau(\mathbf{c_1},\mathbf{c_2})=|\{(i,j)\in[n]^2\mid i\neq j, c_1(i)<c_1(j), c_2(i)>c_2(j)\}|$$

(Recall that $[n]=\{1,2,\ldots,n\}$.) That is, the distance counts pairs of indices where the entries in the two permutations are oppositely ordered. For example, for $q=3$ and $e=2$, let $\mathbf{c_1}=(2,0,0,1,2,1)$ and $\mathbf{c_2}=(1,2,0,0,2,1)$. Then, the pairs of oppositely-ordered indices are at $(2,1),(2,4)$, and $(2,6)$, so that $d_\tau(\mathbf{c_1},\mathbf{c_2})=3$.

Additionally, $d_\tau(\mathbf{c_1},\mathbf{c_2})$ is at least the smallest number of transpositions of adjacent el-

ements required to transform $\mathbf{c_1}$ into $\mathbf{c_2}$. (In the permutation case, where $q = n$, the two concepts are equivalent: the Kendall tau distance $d_\tau(\mathbf{c_1}, \mathbf{c_2})$ is *exactly* equal to the smallest number of transpositions of adjacent elements required to transform $\mathbf{c_1}$ into $\mathbf{c_2}$.)

Although $D_{n,q}$ is not a group, we still refer to the permutation

$$\mathbf{e} = (0, 0, \ldots, 0, 1, \ldots, 1, \ldots, q-1)$$

as the identity multiset permutation. The largest possible $d_\tau$ distance is $e^2\binom{q}{2}$. This occurs when the order of the elements in the identity $\mathbf{e}$ is exactly reversed, such as $(0, 0, 1, 1, 2, 2)$ and $(2, 2, 1, 1, 0, 0)$, which have distance 12.

We define an inversion in a multiset permutation $\mathbf{c}$ as a pair of indices $(i, j) \in [n]^2$ such that $i < j$ and $c(i) > c(j)$. For example $\mathbf{c} = (1, 0, 2, 0, 2, 1)$ has 5 inversions at index pairs $(1, 2), (1, 4), (3, 4), (3, 6)$, and $(5, 6)$.

Next, we introduce the notion of the inversion vector. Again, this is a concept typically applied to the elements of $S_n$ which can be easily extended to multiset permutations. The inversion vector essentially keeps track of the number of inversions involving the various elements of a multiset permutation. There are several possible definitions. We generalize the definition in [MBZ13].

**Definition 7.** *For a permutation $\mathbf{c} \in D_{n,q}$, define the inversion vector $\mathbf{x_c}$ such that*

$$x_{\mathbf{c}}((i-1)e + a) = \left|\left\{(j, b), j \in [q-1], b \in [e] \mid j < i+1, c_b^{-1}(j) > c_a^{-1}(i+1)\right\}\right|,$$

*for $i \in [q-1]$ and $a \in [e]$.*

Here, $c_a^{-1}(j)$ refers to the position of the $a$th entry with value $j$ in $\mathbf{c}$, so that for $\mathbf{c} = (1, 0, 2, 0, 2, 1)$, $c_1^{-1}(2) = 3$ and $c_2^{-1}(2) = 5$.

In words, the entry $x_{\mathbf{c}}((i-1)e+a)$, where $i \in \{1, 2, \ldots, q-1\}$, is the number of inversions in $\mathbf{c}$ in which the $a$th $i$ value is the first entry. We illustrate this idea with an example. Take

$\mathbf{c} = (3, 2, 0, 0, 1, 2, 3, 1)$. There are no inversions beginning with the first 1 (at position 5), so the first element of $\mathbf{x_c}$ is 0. The second 1 (at position 8) also is not the first element in any inversion, so the second element of $\mathbf{x_c}$ is also 0. There are 4 inversions starting with the first 2 in $\mathbf{c}$: the inversions at positions $(2, 3)$, $(2, 4)$, $(2, 5)$, and $(2, 8)$, so $x_{\mathbf{c}}((2-1)\times 2+1) = x_{\mathbf{c}}(3) = 4$, and so on. We have that $\mathbf{x_c} = (0, 0, 4, 1, 6, 1)$.

Note that the length of $\mathbf{x_c}$ is $(q-1)e = n - e$. Also, there are at most $e(i-1)$ elements smaller than $i$ located to the right of it in the permutation $\mathbf{c}$, so $x_{\mathbf{c}}((i-1)e+a) \in [0, e(i-1)]$ so that

$$\mathbf{x_c} \in \underbrace{[0, e] \times \ldots \times [0, e]}_{e \text{ times}} \times \ldots \times \underbrace{[0, (q-2)e] \times \ldots \times [0, (q-2)e]}_{e \text{ times}}.$$

We refer to this space as $G_n$.

It is well known that there is a bijection between the spaces $D_{n,q}$ and $G_n$. That is, there exists a bijection between our codewords, elements in $D_{n,q}$, and their corresponding inversion vectors. It is possible to recover the original permutation $\mathbf{c}$ from its inversion vector $\mathbf{x_c}$. Furthermore, we may introduce the inversion vector $L_1$ distance

$$d(\mathbf{x_{c_1}}, \mathbf{x_{c_2}}) = \sum_{i=1}^{n-e} |x_{\mathbf{c_1}}(i) - x_{\mathbf{c_2}}(i)|.$$

We have the following lemma, which relates the Kendall tau distance $d_\tau$ and the inversion vector distance:

**Lemma 4.** *If $\mathbf{c_1}, \mathbf{c_2} \in D_{n,q}$ are codewords, and $\mathbf{x_{c_1}}, \mathbf{x_{c_2}}$ are the respective inversion vectors,*

$$d_\tau(\mathbf{c_1}, \mathbf{c_2}) \geq d(\mathbf{x_{c_1}}, \mathbf{x_{c_2}}).$$

*Proof.* We begin with the permutation $\hat{\mathbf{c}}_1$ which has a single transposition of adjacent elements but is otherwise the same as $\mathbf{c_1}$. Then, it is clear that $d_\tau(\mathbf{c_1}, \hat{\mathbf{c}}_1) = 1$. We show that $d(\mathbf{x_{c_1}}, \mathbf{x_{\hat{c}_1}}) = 1$.

Let the transposition be at locations $i$ and $i+1$ (for $i \leq i < n$). Let $c_1(i) = \alpha$ and

$c_1(i+1) = \beta$. We only consider the case $\alpha < \beta$, as, by definition, $\alpha \neq \beta$, and the $\alpha > \beta$ case is symmetric. When forming $\mathbf{x}_{\hat{\mathbf{c}}_1}$, no entries other than the ones corresponding to $\alpha$ and $\beta$ can change. The number of inversions in which $\alpha$ is the first element does not change. Only the entry corresponding to $\beta$ at location $i+1$ can be increased by 1 (from the additional inversion formed by the pair $(\beta, \alpha)$ now at positions $(i, i+1)$). Thus the distance $d(\mathbf{x}_{\mathbf{c_1}}, \mathbf{x}_{\hat{\mathbf{c}}_1}) = 1$.

On the other hand, $d(\mathbf{x}_{\mathbf{c_1}}, \mathbf{x}_{\hat{\mathbf{c}}_1}) = 1$ only implies that $d_\tau(\mathbf{c_1}, \hat{\mathbf{c}}_1) \geq 1$. Since $d_\tau$ is at least the number of transpositions of adjacent elements changing one permutation to another, we may repeatedly apply the single transposition case above to conclude that $d_\tau(\mathbf{c_1}, \mathbf{c_2}) \geq d(\mathbf{x}_{\mathbf{c_1}}, \mathbf{x}_{\mathbf{c}_2})$. □

Thus, a code over the space of inversion vectors that corrects $t$ additive errors of weight $l$ will also correct the corresponding errors over the space $D_{n,q}$. In [BM10], Barg and Mazumdar demonstrated how such a code over the space of inversion vectors can be found. We quote the following theorem, which relies on Theorem 3 from the previous section. Denote the set of integers modulo $n$ by $\mathbb{Z}_n$.

**Theorem 4.** *Let $m = \frac{s^{v+1}-1}{s-1}$ and set $v$ to be $t$, the maximum number of errors. Then, generate $s+1$ integers $d_0 = 0, d_1, \ldots d_s$ according to Theorem 3. Take $h_i = d_{i-1} + \frac{t-1}{2}m$ for $i \in [s+1]$ if $t$ is odd and $h_i = d_{i-1} + \frac{t}{2}m$ for $i \in [s+1]$ if $t$ is even. Let $m_t = t(t+1)m$ if $t$ is odd and $m_t = t(t+2)m$ if $t$ is even.*

*Then, for any error vector $\mathbf{e}$ in $\mathbb{Z}^{s+1}$ with $t$ or fewer errors, the sums $\sum_{i=1}^{s+1} e_i h_i$ are all distinct and non-zero modulo $m_t$.*

Now we may define the code $C$ by

$$C = \left\{ \mathbf{x} \in \mathbb{Z}^{s+1} \mid \sum_{i=1}^{s+1} h_i x_i \equiv 0 \bmod m_t \right\}.$$

From Theorem 4, $C$ corrects $t$ additive errors of weight $l$ over $Z_{m_t}^{s+1}$. Our initial goal was to construct a code over the space of inversion vectors $G_n$ that corrects $t$ additive errors. Let

$s+1 = n-e$. Following [BM10], we note that $G_n$ is a subset of $\mathbb{Z}_n^{n-e}$, which is itself a subset of $\mathbb{Z}_{m_t}^{n-e}$. Furthermore, since $C$ is a group code with respect to addition modulo $m_t$, we may take an appropriate coset to form our desired additive error-correcting code over $G_n$.

Thus, we see that there exist $(t, l)$-DTEC codes using the balanced codeword approach as well. We conclude that it is possible to develop dynamic thresholding coding constructions that correct (a fixed number of) limited-magnitude errors in either implementation.

Further details regarding codes based on Theorem 4 above are discussed in [MBZ13]. In this work, a number of code constructions for rank modulation are derived from transformations of traditional Hamming-error correcting codes. Such codes include codes correcting a fixed number of limited-magnitude errors, such as those discussed above, and others where the number of errors corrected is a fraction of the codeword length. Through techniques similar to those previously described, it is possible to generalize these codes to codes over multiset permutations, producing several more flavors of balanced DTEC codes.

### 2.4.3 $l$-DT Error-Correcting Codes

Finally, we wish to correct any number of magnitude $l$ errors (the case where $t \leq n$). Codes that do so for the asymmetric channel have been studied in [CSBB10], [EB10], and [AAKT04]. We develop such a code to be used in conjunction with dynamic thresholds. We refer to this code as an $l$-dynamic thresholding error-correcting code, or an $l$-DTEC code.

The $l$-DTEC constructions will be based on the metadata implementation approach. Again, we assume that the metadata has been made available to the decoder with no error.

We begin by defining an auxilliary distance function which will be useful for defining our construction. The *permutation distance* $d_\pi(\mathbf{c_1}, \mathbf{c_2})$ is a cycle-based measure of difference between codewords that are permutations of each other. If a cycle $(a_1, a_2, \ldots, a_k)$ in a permutation is such that $|a_i - a_{i+1}| \leq l$ for all $i < k$ and $|a_k - a_1| \leq l$, it will be called an $l$-cycle. A permutation that is the product of cycles with length at most $l$ is referred to as an $l$-permutation. Let the function $g$ be defined so that $g(\pi)$ is the smallest nonnegative

integer $l$ such that $\pi$ is an $l$-permutation. Then,

**Definition 8.** *For* $\mathbf{c_1}, \mathbf{c_2} \in F$*, the permutation distance* $d_\pi$ *is given by*

$$d_\pi(\mathbf{c_1}, \mathbf{c_2}) = \begin{cases} \infty & \text{if } \mathbf{c_2} \text{ is not a permutation of } \mathbf{c_1}, \\ g(\pi) & \text{if } \mathbf{c_2} = \pi(\mathbf{c_1}) \text{ is a permutation of } \mathbf{c_1}. \end{cases}$$

Essentially, $\mathbf{c_1}$ and $\mathbf{c_2}$ are at $d_\pi$ distance $l$ if the longest cycle required to turn $\mathbf{c_1}$ into $\mathbf{c_2}$ has length $l$. If no such cycle exists, that is, if $\mathbf{c_1}$ and $\mathbf{c_2}$ are not permutations of one another, the $d_\pi$ distance is $\infty$.

A code of minimum $d_\pi$ distance of $l+1$ corrects any number of errors limited to magnitude $l$. The following construction achieves this minimum $d_\pi$ distance by selecting a single codeword from each set of $l$-permutations.

*Construction 3*: Let $C$ be defined as

$$C = \{\mathbf{x} \in F^n \mid 0 < x_i - x_j \leq l \implies i > j\}.$$

Then, $C$ is a $l$-DTEC code.

Note that this approach is effectively the dual of that of the previous section. When using balanced codes, *all* codewords are permutations of one another. In our $l$-DTEC constructions, *no* codewords are permutations of one another. This is a consequence of choosing between limited or unlimited number of errors in the channel model.

We introduce a natural decoding technique, which we refer to as $\tau$-decoding. First, we comment on the name of the scheme. Although we defined Construction 3 in terms of the distance $d_\pi$, which measures the size of the largest cycle required to turn one permutation into another, the $\tau$-decoder will recover the original codeword transposition by transposition. That is, the decoder will, by stages, reduce the Kendall tau distance between the received permutation and the original codeword.

The $\tau$-decoder operates in the following way: if the decoder finds $y_i$ and $y_j$ in the received

sequence $\mathbf{y}$ with $i < j$ and $0 < y_i - y_j \leq l$, the values of $y_i$ and $y_j$ will be exchanged. The decoder repeats this process as long as such pairs can be found. For example, consider the codeword 12345, where $n = 5$, $q = 6$, and $l = 2$. Assume the errors are

$$1 \to 3,\ 2 \to 1,\ 3 \to 5,\ 4 \to 2,\ 5 \to 4.$$

The received codeword is 31524. Repeatedly applying the decoding rule, we have the progression

$$31524 \to 13524 \to 13425 \to 13245 \to 12345.$$

As described above, $\tau$-decoding reduces the Kendall tau distance $d_\tau$ between the original codeword and the received codeword. Recall that the distance $d_\tau$ is defined as the number of inversions among two permutations $\pi$ and $\sigma$, where an inversion is a pair $(i, j)$ with $i < j$, with either $\pi(i) < \pi(j)$ and $\sigma(i) > \sigma(j)$, or, $\pi(i) > \pi(j)$ and $\sigma(i) < \sigma(j)$ ([Ken38]). We have that:

**Theorem 5.** *When errors are limited to magnitude $l$ and dynamic thresholds and the $l$-DTEC code $C$ in Construction 3 are used, $\tau$-decoding recovers the original codeword.*

*Proof.* Using dynamic thresholding, errors will be part of cycles. Let the cells involved be $x_{p_1}, \ldots, x_{p_k}$ where $p_1 < \ldots < p_k$. By definition of $C$, the cell values also satisfy $a_1 < a_2 < \ldots < a_k$. The error cycle permutes the $a_i$'s, so that $x_{p_1}, \ldots, x_{p_k}$ take on the values $\pi(a_1), \ldots, \pi(a_k)$, where $\pi$ is the permutation induced by the error cycle. Let $\mathbf{y}_\alpha$ be the current output after $\alpha$ iterations of the decoding process. We will evaluate the distance $d_\tau(\mathbf{y}_\alpha, \mathbf{x})$ after each iteration. If $d_\tau(\mathbf{y}_\alpha, \mathbf{x}) = 0$, we have recovered the original codeword.

The decoding rule at each iteration finds a particular inversion $(a_i, a_j)$ with $i < j$ and $\pi(a_i) > \pi(a_j)$ and exchanges their values. We compute the distance before and after one decoding iteration, i.e., $d_\tau(\mathbf{y}_\alpha, \mathbf{x})$ and $d_\tau(\mathbf{y}_{\alpha+1}, \mathbf{x})$. The only possible pairs that might change (become an inversion or stop being an inversion) after application of the decoding rule are those which have $\pi(a_i)$ or $\pi(a_j)$ as members. We count the number of such pairs. Consider

the following (disjoint) sets:

$$\begin{aligned}
Y_1 &= \{k \mid k < i, \pi(a_k) > \pi(a_i) > \pi(a_j)\},\\
Y_2 &= \{k \mid k < i, \pi(a_i) > \pi(a_k) > \pi(a_j)\},\\
Z_1 &= \{k \mid i < k < j, \pi(a_k) > \pi(a_i) > \pi(a_j)\},\\
Z_2 &= \{k \mid i < k < j, \pi(a_i) > \pi(a_k) > \pi(a_j)\},\\
Z_3 &= \{k \mid i < k < j, \pi(a_i) > \pi(a_j) > \pi(a_k)\},\\
W_1 &= \{k \mid j < k, \pi(a_i) > \pi(a_j) > \pi(a_k)\},\\
W_2 &= \{k \mid j < k, \pi(a_i) > \pi(a_k) > \pi(a_j)\}.
\end{aligned}$$

The entries in the permutation where these sets are located:

$$\overbrace{\pi(a_1),\ldots,}^{Y_1,Y_2}\pi(a_i),\underbrace{\pi(a_{i+1}),\ldots,\pi(a_{j-1})}_{Z_1,Z_2,Z_3},\pi(a_j),\overbrace{\ldots,\pi(a_k)}^{W_1,W_2}.$$

It is easy to see how the members of these sets give rise to inversions. For example, if $k \in W_1$, then, $i < j < k$ and $\pi(a_i) > \pi(a_j) > \pi(a_k)$, so $(\pi(a_i), \pi(a_k))$ and $(\pi(a_j), \pi(a_k))$ are inversions. It is clear that each element of each set contributes to either one or two inversions.

Say that there are $M$ pairs that do not involve $\pi(a_i)$ or $\pi(a_j)$. Then, counting the contribution of the pair $(\pi(a_i), \pi(a_j))$, we have the following value for $d_\tau(\mathbf{y}_\alpha, \mathbf{x})$:

$$d_\tau(\mathbf{y}_\alpha, \mathbf{x}) = M + 1 + 2|Y_1| + |Y_2| + |Z_1| + 2|Z_2| + |Z_3| + 2|W_1| + |W_2|.$$

Now, after application of the decoding rule, the pair $(\pi(a_i), \pi(a_j))$ becomes $(\pi(a_j), \pi(a_i))$, and is no longer an inversion. Moreover, $Z_2$ does not contribute to any inversions, since $i < j < k$ and $\pi(a_j) < \pi(a_k) < \pi(a_i)$. Then,

$$d_\tau(\mathbf{y}_{\alpha+1}, \mathbf{x}) = M + 2|Y_1| + |Y_2| + |Z_1| + |Z_3| + 2|W_1| + |W_2|.$$

Thus, we have that $d_\tau(\mathbf{y}_\alpha, \mathbf{x}) > d_\tau(\mathbf{y}_{\alpha+1}, \mathbf{x})$. Each decoding step strictly reduces the distance, so there exists a finite $s$ such that $d_\tau(\mathbf{y}_s, \mathbf{x}) = 0$ and we will have recovered the original codeword. □

In the case $l = 1$, this decoder has a simple implementation. The received sequence is first sorted by value. All errors will then be adjacent, and can be corrected in a single pass through the list. The original order is then restored. This algorithm has complexity $O(n \log n)$, the complexity of sorting the list.

We note that we can build an $l$-DTEC code from Construction 3 by including every codeword from an existing $l$-asymmetric error-correcting ($l$-AEC) code, which has minimum distance $l + 1$, as shown in [EB10]. Note that although limited-magnitude errors under dynamic thresholding are not asymmetric, the permutation property of dynamic thresholding errors ensures that $l$-AEC constructions are sufficient to correct $l$-DT errors. For this reason, $l$-DTEC constructions most closely resemble $l$-AEC constructions. However, several additional codewords can also be included in addition to the $l$-AEC codewords, allowing us to maximize the rate under the dynamic thresholding conditions.

We illustrate this idea with an example of a code based on Construction 3. We take $q = 3$, limited magnitude $l = 1$, and $n = 3$. An 1-AEC code of length 3 has minimum distance of 2, requiring codewords to exclusively use symbols from $\{0, 2\}$. In our case, we may add several codewords that are ordered the correct way. Our codewords are then:

$$(0,0,0), (0,0,2), (0,2,0), (2,0,0), (0,2,2), (2,0,2), (2,2,0),$$
$$(2,2,2), (0,0,1), (0,1,1), (1,1,1), (0,1,2), (1,1,2), (1,2,2).$$

We see that the first 8 codewords are from the 1-AEC code with minimum distance 2, but we may add 6 additional codewords that maintain the correct order.

Computing the sizes of the codebooks in Construction 3 can be challenging. We give the following exact expression for the case $q = 3$:

**Theorem 6.** *For $q = 3$, $l = 1$ and general $n$, the number of codewords generated by Construction 3 is $2^n + \binom{n+1}{2}$.*

*Proof.* If a codeword does not contain any 1s, each digit may be 0 or 2, for a total of $2^n$ such codewords. If a codeword does contain a 1, it is clear that all its 0s must be to the left of all its 1s. Similarly, all 2s must be to the right of all 1s. Thus, there is a left and a right boundary (not in the same position, since there are a positive number of 1s.) There are $n + 1$ positions to place the two boundaries, so that we have a total of $\binom{n+1}{2}$ codewords containing 1s. □

Not including the metadata digits, the rate of such a code is $\log_3(2^n + \binom{n+1}{2})/n$. When our codes are in the finite-length regime, we get the benefit of an additional $\binom{n+1}{2}$ codewords versus directly combining dynamic thresholding and an existing 1-AEC code with minimum distance 2.

We note that the construction above is only suitable for higher field sizes $q$. Furthermore, asymptotically, the size of the code is $\lceil \frac{q}{l+1} \rceil^n$, which is the size of the largest $l$-AEC code [EB10]. Therefore, our construction is superior to $l$-AEC constructions for finite-length codes with $q \geq 4$. These conditions match the requirements of MLC non-volatile memories.

# CHAPTER 3

# Constrained Rank Modulation

## 3.1 Introduction

In this chapter, we introduce the notion of constrained coding in the rank modulation scheme. We begin with some details on rank modulation and the inter-cell coupling issue. Constraints are given which help reduce this problem. In particular, we analyze the single neighbor $k$-constraint, where adjacent cells in the permutation differ by at most $k$.

The rank modulation scheme for NVM memories was first introduced in [JMB09]. In rank modulation, information in not stored in individual cells, but rather in entire blocks. The level of any particular cell does not matter; information is represented by the rankings of the charge levels in all the cells in a block. Ties between charge levels are not allowed, so there are $n!$ possible rankings of cells in a block of length $n$. Thus, rank modulation is a permutation-based scheme.

We give an example of the above idea. Say we have a block of cells $\mathbf{x} = (x_1, x_2, \ldots, x_5)$ of length 5 with charge levels given by $\mathbf{v} = (2.8, 1.5, 2.6, 1.4, 1.2)$. Cell $x_1$ has the largest charge value, followed by cell $x_3$, and so on. The rankings of the charge levels $\mathbf{v}$ induce the permutation $\sigma = (1\ 3\ 2\ 4\ 5)$. Any of the permutations in the symmetric group $S_5$ could be represented by varying the charge levels.

The fundamental advantage of rank modulation is that the scheme resolves the write-asymmetry issue. In the example above, we can reach any other permutation in $S_5$ from $\sigma$

without ever removing charge from a cell. It is enough to add charge to an appropriate subset of cells. For example, should we wish to transition to the permutation $\sigma' = (2\ 3\ 1\ 5\ 4)$, we need only increase the charge level of $x_3$ to 3.0 and the charge level of $x_5$ to 1.45.

Recently, there has been a great deal of interest in rank modulation. A number of variants of the scheme have been introduced, such as local rank modulation, where the permutation is induced by a sliding window over a sequence of charge values [ELSB11b], [ELSB11a], and bounded rank modulation, where permutations are allowed to overlap [ZJ10]. Similarly, in partial rank modulation, only the largest $r$ cell levels are considered for information representation [ZJB09]. Error correction for rank modulation has been explored in [BM10], [MBZ13], [ZJB12b], and [ZPJ10].

We note that the inter-cell coupling problem is not resolved in rank modulation. Recall that inter-cell coupling is a parasitic capacitance experienced by physically adjacent cells. When a cell contains a large charge level, the level of a neighboring cell may be inadvertently brought up. To resolve this problem, various types of constrained coding have been introduced, such as in [BB11]. However, this work deals with cells which store information according to their absolute charge levels, unlike the rank modulation scheme. In fact, the effects of inadvertently increasing the charge level in a neighboring cell may be even more significant in rank modulation: an increase in one cell can dramatically change the resulting permutation.

We illustrate this idea with an example: again, take the block of cells $\mathbf{x}$ with charge levels $\mathbf{v} = (2.8, 1.5, 2.6, 1.4, 1.2)$ and induced permutation $\sigma = (1\ 3\ 2\ 4\ 5)$. Say we wish to write the permutation $\sigma' = (2\ 4\ 3\ 1\ 5)$ to our block. Doing so requires a single operation: increasing the charge level in cell $x_4$, with current charge level 1.4. This level must be increased beyond 2.8, the current largest charge level in the block $\mathbf{x}$. However, pushing $x_4$ to 3.0 can cause inter-cell coupling effects, resulting in an increase in the charge in $x_5$ to, for example, 1.8. Now we have $\mathbf{v}' = (2.8, 1.5, 2.6, 3.0, 1.8)$, with $\sigma = (2\ 5\ 3\ 1\ 4)$. We have an error: the 2nd and 5th positions are incorrect.

In order to avoid situations such as that in the above scenario, we explore the notion of constrained rank modulation schemes. We seek codes on permutations which meet constraints between the relative rankings of neighboring cells. Such codes retain the advantages of rank modulation while reducing the effects of inter-cell coupling. The remainder of this chapter is organized as follows: In Section 3.2, we introduce the constraints and necessary notation. In Section 3.3, we examine some aspects of the single neighbor constraint. We derive bounds and an efficient construction. We also analyze the asymptotic behavior of this constraint. Finally, we examine two special cases where the constraint parameter $k$ involves constants.

## 3.2 Types of Constraints

We start by giving some useful notation. Define $[n]$ as the set $\{1, 2, \ldots, n\}$ and let $S_n$ be the symmetric group of permutations over $[n]$. We use the one line notation for permutations so that $\sigma = (\sigma_1\ \sigma_2\ \ldots\ \sigma_n)$. For example, we have the permutation $(2\ 1\ 4\ 3\ 5) \in S_5$.

Our first constraint is the single neighbor $k$-constraint:

**Definition 9.** *A permutation $\sigma \in S_n$ satisfies the single neighbor $k$-constraint if $|\sigma_i - \sigma_{i+1}| \leq k$ for all $1 \leq i \leq n-1$.*

For example, the permutation $(3\ 1\ 2\ 4\ 5) \in S_5$ meets the 2-constraint, but not the 1-constraint. Note how this constraint resolves the coupling issue described earlier. Differences between the rankings of adjacent cells are always limited, so that charge differences between adjacent cells are also limited. Any transition undergone by a block must be from one constrained permutation to another, preserving these difference limitations.

We define $A_{n,k} = \{\sigma \in S_n \mid \sigma \text{ satisfies the } k\text{-constraint}\}$ as the set of permutations on $n$ elements that satisfy the single neighbor $k$-constraint. Of course, $A_{n,k} \subseteq S_n$. In particular, if $n \leq k+1$, $A_{n,k} = S_n$. Consider a set of $k$-constrained permutations $C \subseteq A_{n,k}$. Then, we

define the rate of $C$ as

$$\mathcal{R}(C) = \frac{\ln |C|}{\ln n!}.$$

We define the capacity of $k$-constrained codes as

$$\mathcal{C}(k) = \lim_{n\to\infty} \frac{\ln |A_{n,k}|}{\ln n!}.$$

The single neighbor constraint will be the main focus of this chapter. However, there are other potentially useful constraints. We give two examples:

**Definition 10.** *A permutation $\sigma \in S_n$ violates the two neighbor $k$-constraint if $|\sigma_{i-1} - \sigma_i| > k$ and $|\sigma_{i+1} - \sigma_i| > k$ for some $2 \leq i \leq n-1$.*

Here, we avoid those permutations where a small element is sandwiched between two large elements. Note that this is a looser constraint in comparison to the single neighbor constraint. This is because meeting the single neighbor $k$-constraint guarantees meeting the two neighbor $k$-constraint, but meeting the two neighbor $k$-constraint does not guarantee meeting the single neighbor $k$-constraint.

Our constraints so far apply to one-dimensional sequences. However, in NVMs the cells are organized as a large two-dimensional array. Consider such an array of size $m \times p$ and say the array is filled with permutations on $n = mp$. Then, we define the two-dimensional $k$-constraint in the following way:

**Definition 11.** *The two-dimensional $k$-constraint is met if the value in the induced permutation $\sigma \in S_n$ of the cell $x_{i,j}$ is within $k$ of the values of the cells $\{x_{a,b} \mid a \in \{i-1, i, i+1\}, b \in \{j-1, j, j+1\}\}$ for all $2 \leq i \leq m-1$ and $2 \leq j \leq p-1$.*

With this constraint, the neighbors of cell $x_i$ include the cells above and below it, and not just those cells to the immediate left and right.

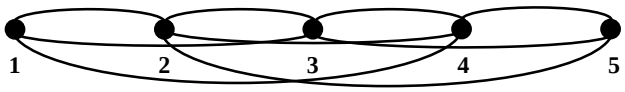


Figure 3.1: The path-scheme $P(5, \{1, 2, 3\})$.

## 3.3 Single Neighbor Constraint

We proceed to analyze the previously introduced single neighbor $k$-constraint.

### 3.3.1 General Bounds

We start with a general bound on the sizes of single neighbor $k$-constrained rank modulation codes. First, as in [AK08], we represent permutations as paths in structures called path-schemes. Define the path-scheme $P(n, M)$ as the graph $G = (V, E)$ with $V = [n]$ and $E = \{(x, y) \mid |x - y| \in M\}$. An example is shown in Figure 3.1. The example shows the case where $n = 5$ and the set $M = \{1, 2, 3\}$.

We define $G_{k,n}$ as $P(n, \{1, 2, \ldots, k\})$. In [AK08], it was shown that there is a bijection between the set of directed Hamiltonian paths in $G_{k,n}$ and the set of $k$-constrained permutations. Using this fact, we show the general bound below, which is an improvement on that given in [AK08].

**Theorem 7.** *The number of $k$-constrained permutations $|A_{n,k}|$ satisfies*

$$\left(\frac{(k+1)!}{2}\right)^{\lfloor \frac{n}{k} \rfloor - 1} \leq |A_{n,k}| \leq \left(\frac{k(k+1)}{2}\right)^{\lfloor \frac{n}{k} \rfloor - 1} (2k-2)^{n - \lfloor \frac{n}{k} \rfloor + 1}.$$

*Proof.* We count directed Hamiltonian paths on $G_{k,n}$. Split up $[n] = \{1, 2, \ldots, n\}$ into intervals of length $k$ in the following way:

$$[1, k], [k+1, 2k], [2k+1, 3k], \ldots, \left[\left(\left\lfloor \frac{n}{k} \right\rfloor - 1\right) k, n\right].$$

Each interval has length $k$, except possibly the last interval, which has length at least $k$. Now, consider any Hamiltonian path on $G_{k,n}$. Such a path must include an edge going from a node in the $\ell$th interval to a node in the $(\ell+1)$st interval. If this is not the case and the $\ell$th interval is skipped, there is an edge $(x, y)$ with $|x-y|$ larger than $k$, which is not possible by definition of $G_{k,n}$.

The interval length is $k$, so the number of possible edges between two adjacent intervals is $k(k+1)/2$. There are $\lfloor \frac{n}{k} \rfloor - 1$ adjacent pairs of intervals. Therefore, we have a factor of

$$\left(\frac{k(k+1)}{2}\right)^{\lfloor \frac{n}{k} \rfloor - 1}$$

in our count.

To produce the lower bound, we restrict ourselves to the case where each edge (excluding the interval-crossing edges counted above) remains inside their outgoing node's interval. That is, we count paths where for each interval of length $k$, there is one edge leaving the interval, and all other outgoing edges remain inside it. There are $k-1$ remaining vertices in each interval requiring an edge (and possibly more in the last interval). The edges are selected in $(k-1)!$ ways. Multiplying, we get the expression

$$\left(\frac{k(k+1)}{2}(k-1)!\right)^{\lfloor \frac{n}{k} \rfloor - 1},$$

which gives the left-hand side.

For the upper bound, we place no restriction on the remaining edges. Each node has at most $2k$ possible choices of edges in the path. Two of these are no longer available, due to the interval-crossing edges already selected. Therefore, we let each vertex take on any of the $2k-2$ possible remaining edges. There are $n - (\lfloor \frac{n}{k} \rfloor - 1)$ such vertices, which yields a factor of

$$(2k-2)^{n-(\lfloor \frac{n}{k} \rfloor - 1)}.$$

Multiplying by the interval-crossing factor $(k(k+1)/2)^{\lfloor \frac{n}{k} \rfloor - 1}$ gives the upper bound. $\square$

Note that this general bound is quite rough. In order to get better estimates on the size of $A_{n,k}$, it is necessary to examine particular choices of $k$, which we do in the following sections.

### 3.3.2 An Efficient Construction

In the previous section, we determined a bound on the number of $k$-constrained permutations in $S_n$. Unfortunately, encoding is quite difficult. For example, if we wish to encode a binary sequence in $\{0,1\}^m$ as a $k$-constrained permutation, we would need to use the above bound to determine some $n$ such that $|A_{n,k}| \geq |\{0,1\}^m| = 2^m$, and then generate the entire set of permutations $S_n$, going through each permutation element by element to determine those which are $k$-constrained. Afterwards, some ordering would be imposed on the $k$-constrained permutations and used to perform the mapping.

To resolve this difficulty, we introduce the construction $C_{n,k}$, which forms a $k$-constrained rank modulation code. The sets $C_{n,k}$ will be built recursively, and we will provide an efficient encoding scheme to map (binary) sequences to $k$-constrained permutations in $C_{n,k}$. Furthermore, we will later see that the construction is asymptotically optimal. We begin with the definition of $C_{n,k}$:

*Construction 1*: Let $C_{n,k}$ be defined in the following way:

$$C_{n,k} = \begin{cases} S_n & n \leq k+1 \\ \{f_{n,k}(\sigma) \mid \sigma \in C_{n-1,k}\} & n > k+1, \end{cases}$$

where $n \geq 4, k \geq 2$, and $f_{n,k} : S_{n-1} \to S_n$ is a one-to-many map which takes a permutation of length $n-1$ and places the element $n$ in it (in various positions) to form $\lfloor \frac{k}{2} \rfloor$ distinct $k$-constrained permutations of length $n$. Specifically, $f_{n,k}$ examines the locations to the left and the right of the positions of the values $n-1, n-2, n-3, \ldots, n-k$ in $\sigma$, respectively,

and in that order. The process is stopped when $\lfloor \frac{k}{2} \rfloor$ distinct positions have been found to place $n$ (and thus yield $\lfloor \frac{k}{2} \rfloor$ distinct $k$-constrained permutations of length $n$.)

We give an illustrative example of this technique. Say we wish to encode the binary sequence $(0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0)$ into a 4-constrained permutation using Construction 1. First, $k = 4$ implies that all $5! = 120$ permutations in $S_5$ are available to us, so the first $\lfloor \log_2 120 \rfloor = 6$ bits will be mapped to a permutation of length 5. Any map can be used here; we rely on a lexicographical ordering of permutations such that $(0, 0, 0, 0, 0, 0)$ is mapped to (1 2 3 4 5). Then, the first 6 bits $(0, 1, 1, 0, 1, 0)$ map to the permutation (2 1 3 5 4).

As $k = 4$, $\lfloor \frac{k}{2} \rfloor = 2$, and each subsequent bit is used to determine into which of two positions the next element goes. If the next bit is 0, this element goes in the first available position; if it is 1, it goes in the second such position. The element 6 can be placed to the left or the right of 5 in (2 1 3 5 4) according to the map $f_{6,4}$. Our binary sequence's next bit is 1, so 6 is placed to the right of 5 in the permutation, giving (2 1 3 5 6 4). (If the bit had been 0, the element 6 would have been to the left of 5, yielding (2 1 3 6 5 4).)

The remainder of the binary sequence is $(0, 1, 1, 1, 0)$, which yields, using the same procedure, the sequence of permutations

$$(2\ 1\ 3\ 5\ 6\ 4)$$

$$(2\ 1\ 3\ 5\ 7\ 6\ 4)$$

$$(2\ 1\ 3\ 5\ 7\ 8\ 6\ 4)$$

$$(2\ 1\ 3\ 5\ 7\ 8\ 9\ 6\ 4)$$

$$(2\ 1\ 3\ 5\ 7\ 8\ 9\ 10\ 6\ 4)$$

$$(2\ 1\ 3\ 5\ 7\ 8\ 9\ 11\ 10\ 6\ 4).$$

Therefore, we encode the sequence $(0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0)$ as the 4-constrained permutation (2 1 3 5 7 8 9 11 10 6 4).

Based on the above idea, we provide a formal mapping $\Omega$ from the space of binary se-

quences to $C_{n,k}$, where $n > k+1$. First, let

$$g : \{0,1\}^{\lfloor \log_2 (k+1)! \rfloor} \to S_{k+1}$$

be any injective map from binary sequences to permutations. We also denote the $j$th permutation produced by the map $f_{n,k}(\sigma)$ by $f_{n,k}(\sigma, j)$, for $1 \leq j \leq \lfloor \frac{k}{2} \rfloor$. Then, take any binary sequence $\mathbf{x}$ and write it as $\{\mathbf{x}_1 | \mathbf{x}_2 | \ldots | \mathbf{x}_\ell\}$, where $\mathbf{x}_1 \in \{0,1\}^{\lfloor \log_2 (k+1)! \rfloor}$ and $\mathbf{x}_2, \ldots, \mathbf{x}_\ell \in \{0,1\}^{\lfloor \log_2 \lfloor \frac{k}{2} \rfloor \rfloor}$. With this, we may write

$$\Omega(\mathbf{x}) = f_{n,k}(f_{n-1,k}(\ldots(f_{k+2,k}(g(\mathbf{x_1}), x_2), x_3)\ldots, x_{l-1}), x_\ell).$$

We comment on the advantages of Construction 1. First, the length of the binary sequence and the $k$-constraint immediately provide us with the length $n$ of the permutations necessary for encoding. Second, generating a single $k$-constrained permutation in $S_n$ with $n > k+1$ is very fast: we select any permutation from $S_{k+1}$ and perform at most $2(n-k-1)$ checks while building it to length $n$. This further implies that a small number of permutations can be generated very quickly. Finally, the construction provides a systematic technique to generate exponentially large numbers of $k$-constrained permutations.

Next, we proceed to prove that the process described in Construction 1 indeed yields the required $\lfloor \frac{k}{2} \rfloor$ $k$-constrained permutations at each stage. Before we begin, we explain the use of the term $\lfloor \frac{k}{2} \rfloor$. It turns out to be the largest number $y$ of $k$-constrained permutations that will (each) always yield $y$ $k$-constrained permutations in the next stage.

**Theorem 8.** *The function $f_{n,k}(\sigma)$ in Construction 1 yields $\lfloor \frac{k}{2} \rfloor$ $k$-constrained permutations if $\sigma \in C_{n-1,k}$, $n \geq 4$, and $k \geq 2$.*

*Proof.* The proof is inductive. It relies on a simple argument, but requires careful bookkeeping. Let us say that we have a permutation $\sigma \in C_{n-1,k}$. We must show that, using the map $f_{n,k}$, we may form at least $\lfloor \frac{k}{2} \rfloor$ $k$-constrained permutations from $\sigma$. First, if $n \leq k+1$,

all permutations in $S_n$ are $k$-constrained by definition. Similarly, if $n = k + 2$, the only prohibited positions for $n$ are those adjacent to the element 1 in $\sigma$, leaving $n - 2 \geq \lfloor \frac{k}{2} \rfloor$ positions available for $n$.

Now we focus on $n > k + 2$. Assume that the result is true for $C_{n-1,k}$. Let us refer to the positions of $n-1, n-2, \ldots, n-k$ in $\sigma$ as $s_{n-1}, s_{n-2}, \ldots, s_{n-k}$, respectively. Note that at least one of the positions adjacent to $n-1$ in $\sigma$, $s_{n-1}-1$ and $s_{n+1}+1$, is available to $n$. The smallest possible element adjacent to $n-1$ is $n-k-1$, since $\sigma$ is $k$-constrained. Therefore, at least one element adjacent to $n-1$ is greater than or equal to $n-k$ (or, alternatively, $n-1$ is on the edge of the permutation, so that $n$ can be trivially placed next to it). The element $n$ can be placed between this element and $n-1$. We then have two possibilities:

First, only one position adjacent to $n-1$ is available to $n$. This gives one initial possible permutation, which we call $\sigma_1 \in S_n$. In this case, the other element adjacent to $n-1$ must be $n-k-1$. Now, remove $n-1$ from our permutation $\sigma$, yielding a permutation $\sigma'$ of length $n-2$. Placing $n$ into this permutation is in fact equivalent to placing $n-1$ into $\sigma'$, but with a constraint of $k-1$ rather than $k$. By induction, there are $\lfloor \frac{k-1}{2} \rfloor$ distinct ways to do so. In none of these permutations can we place $n$ where the deleted element $n-1$ was. Doing so would be equivalent to placing $n-1$ in $s_{n-1}$, but, as $n-k-1$ is adjacent to this position, $(n-1)-(n-k-1) > (k-1)$ and the $(k-1)$-constraint is violated. Thus none of these permutations can be identical to $\sigma_1$.

After generating the permutations which include $n$, we restore $n-1$ to its original position. Counting the first permutation $\sigma_1$ and the additional $\lfloor \frac{k-1}{2} \rfloor$ permutations, we have $1 + \lfloor \frac{k-1}{2} \rfloor \geq \lfloor \frac{k}{2} \rfloor$ $k$-constrained permutations.

Next, both positions next to $n-1$ in $\sigma$ are available to $n$. This gives us 2 permutations, $\sigma_1, \sigma_2 \in S_n$. We use similar reasoning. Remove $n-1$ from $\sigma$, yielding $\sigma'$. Placing $n$ in $\sigma'$ is equivalent to placing $n-1$ in $\sigma'$ with a $k-1$ constraint. There are (by induction) $\lfloor \frac{k-1}{2} \rfloor$ ways to do this. Of course, we may not count the permutation which places $n$ in $n-1$'s position $s_{n-1}$ (prior to its removal), as this is included in the first two permutations $\sigma_1, \sigma_2$. Counting

these initial 2 permutations, we have $2 + (\lfloor \frac{k-1}{2} \rfloor - 1) \geq \lfloor \frac{k}{2} \rfloor$ permutations, as desired. $\square$

Now, as $f_{n,k}$ produces $\lfloor \frac{k}{2} \rfloor$ permutations for each stage (beyond the initial $n \leq k+1$ stages), $|C_{n,k}| = (k+1)! \lfloor \frac{k}{2} \rfloor^{n-k-1}$. Since $C_{n,k} \subseteq A_{n,k}$, we have the following lower bound:

**Theorem 9.** *The number of $k$-constrained permutations $|A_{n,k}|$ satisfies*

$$|A_{n,k}| \geq (k+1)! \left\lfloor \frac{k}{2} \right\rfloor^{n-k-1}.$$

This lower bound is better than the left-hand side of Theorem 1. It will be particularly useful in the asymptotic analysis performed below.

### 3.3.3 Asymptotic Analysis

In this section, we compute the capacity of $A_{n,k}$. We start with the lower bound from the previous section. We have that $|A_{n,k}| \geq (k+1)! \lfloor \frac{k}{2} \rfloor^{n-k-1}$. Then, taking logarithms,

$$\begin{aligned}
&\ln |A_{n,k}| \\
&\quad \geq \quad \ln(k+1)! + (n-k-1) \ln \left\lfloor \frac{k}{2} \right\rfloor \\
&\quad \geq \quad \ln(k+1)! + (n-k-1) \ln \frac{k-1}{2} \\
&\quad = \quad (k+1) \ln(k+1) + (n-k-1) \ln \frac{k-1}{2} - g(k),
\end{aligned}$$

where $g(k)$ is an error term which is roughly $k+1-O(\ln(k+1))$[1]. Continuing, as $k+1 \geq \frac{k-1}{2}$, we write

$$\begin{aligned}
\ln |A_{n,k}| &\geq n \ln \frac{k-1}{2} - g(k) \\
&= n \ln(k-1) - n \ln 2 - g(k).
\end{aligned}$$

[1] $f(x) = O(g(x))$ if there exists $M > 0$ and $x_0$ such that $|f(x)| \leq M|g(x)| \; \forall x \geq x_0$. Similarly, $f(x) = \Theta(g(x))$ if there exists $M_1, M_2 > 0$ and $x_0$ such that $M_1|g(x)| \leq |f(x)| \leq M_2|g(x)| \; \forall x > x_0$.

Next, to compute the capacity $\mathcal{C}(k)$, we use Stirling's approximation $\ln n! = n \ln n - n + O(\ln n)) \leq n \ln n$, for sufficiently large $n$. Then,

$$\begin{aligned} \frac{\ln |A_{n,k}|}{\ln n!} &\geq \frac{n \ln(k-1) - n \ln 2 - g(k)}{n \ln n} \\ &= \frac{\ln(k-1)}{\ln n} - \frac{\ln 2}{\ln n} - \frac{g(k)}{n \ln n}. \end{aligned}$$

Now take $k = \Theta(n^\epsilon)$, where $0 \leq \epsilon \leq 1$. Since the right two terms go to zero when we take the limit, we may write

$$\begin{aligned} \mathcal{C}(k) &\geq \lim_{n \to \infty} \frac{\ln(\Theta(n^\epsilon))}{\ln n} \\ &= \epsilon. \end{aligned}$$

Now we compute an upper bound for $\mathcal{C}(k)$. We start with $|A_{n,k}| \leq 2(2k)^n$, which is given in [AK08]. Taking logarithms,

$$\begin{aligned} \frac{\ln |A_{n,k}|}{\ln n!} \leq \frac{\ln 2}{\ln n!} + \frac{n \ln 2}{\ln n!} + \frac{n \ln k}{\ln n!} \\ \leq \frac{\ln 2}{n \ln n - n} + \frac{n \ln 2}{n \ln n - n} + \frac{n \ln k}{n \ln n - n}. \end{aligned}$$

The first two terms on the right-hand side will go to zero when taking the limit $n \to \infty$, so we may write:

$$\mathcal{C}(k) = \lim_{n \to \infty} \frac{\ln |A_{n,k}|}{\ln n!} \leq \lim_{n \to \infty} \frac{n \ln k}{n \ln n - n} = \lim_{n \to \infty} \frac{\ln k}{\ln n - 1}.$$

Taking $k = \Theta(n^\epsilon)$ and using L'Hopital's rule on the expression yields $\mathcal{C}(k) \leq \epsilon$. Therefore, we may state

**Theorem 10.** *If $k = \Theta(n^\epsilon)$, where $0 \leq \epsilon \leq 1$, the capacity of $k$-constrained rank modulation codes is $\mathcal{C}(k) = \epsilon$.*

We comment on this result. If $k$ is some constant $c$, then $\epsilon = 0$ and the capacity $\mathcal{C}(k)$ is 0.

Clearly, such a restrictive constraint is not useful in the context of rank modulation. On the other hand, if we need only avoid the largest differences among cells, we may set $k = n - c$, where $c$ is a constant. Then, $\epsilon = 1$ and the capacity is 1. We further examine these two cases in the following section. We also note that the proof of Theorem 3 implies that $C_{n,k}$ has asymptotic rate $\epsilon$, so it is asymptotically optimal in addition to being of practical use.

### 3.3.4 Constant Case I

We begin with the case where $k = c$ is a small constant. Most existing work with $k$-constrained permutations involves this case. For example, there are several known results regarding $c = 2$ [AK08]. The generating function for $|A_{n,2}|$ follows:

$$G_2(x) = \frac{1 - 2x + 2x^2 + x^3 - x^5 + x^6}{(1 - x - x^3)(1 - x)^2}.$$

A technique known as the transfer matrix method is used to count the members of $A_{n,2}$. It is possible to derive similar generating functions for $c > 2$. However, the transfer matrix method, which requires computing the eigenvalues of a matrix whose dimensions grow as $nk$, quickly becomes unwieldy. We state a different fact regarding the growth rate of $A_{n,2}$.

**Theorem 11.** $\lim_{n\to\infty} |A_{n+1,2}|/|A_{n,2}| = \gamma_2$, *where* $\gamma_2 = 1.46557...$ *is the real root of the polynomial* $x^3 - x^2 - 1$.

*Proof.* The denominator of the generating function for $|A_{n,2}|$ contains the term $1 - x - x^3$. Then, for sufficiently large $n$, $|A_{n,2}|$ satisfies the recursion $|A_{n+3,2}| = |A_{n+2,2}| + |A_{n,2}|$. The characteristic polynomial for this recursion is $x^3 - x^2 - 1$. It has $\gamma_2$ as its only real root, so that $|A_{n,2}| = c\gamma_2^n$ for some constant $c > 0$ and large $n$. □

Numerical results show that a similar result applies for $k$ which are constants larger than 2. We believe that for such cases the growth factor $\gamma_k$ is similarly a real root of a polynomial with degree roughly $k$. Certainly, we know from Construction 1 that $\gamma_k \geq \lfloor \frac{k}{2} \rfloor$. It is an interesting question how large the error $\gamma_k - \lfloor \frac{k}{2} \rfloor$ is. For example, $\gamma_2 - 1 \approx 0.466$.

### 3.3.5 Constant Case II

Now we examine the case where $k = n-c$ and $c \geq 1$ is some constant. This case is particularly meaningful for rank modulation, since only the largest possible differences among consecutive elements are constrained. We compute $|A_{n,n-c}|$ through inclusion-exclusion type counting arguments.

We begin by defining a pattern as a consecutive sequence of permutation elements, where each two consecutive elements violate the $k$-constraint. We call a pattern of length $\ell$ an $\ell$-pattern. For example, (1 5 9 15) is a 4-pattern for $k = 3$. In general, the value of $k$ will be clear from the context.

The largest adjacent difference possible in elements of $S_n$ is $n-1$, so the smallest non-trivial case is $c = 2$. Then, the only inadmissible 2-patterns of consecutive elements are $(1\ n)$ and $(n\ 1)$. Now, for each of these there are $n-1$ positions to place them (as a unit) in the permutation over $[n]$, and $(n-2)!$ ways to permute the remaining elements $\{2, 3, \ldots, n-1\}$. Thus there are $2(n-1)!$ permutations which do not meet the constraint, so that $|A_{n,n-2}| = n! - 2(n-1)! = (n-2)(n-1)!$.

The next case, $c = 3$, is instructive: there are several 2-patterns to avoid: $(1\ n)$, $(1\ n-1)$, $(2\ n)$, and their reflections $(n\ 1), (n-1\ 1)$, and $(n\ 2)$. However, by counting how many permutations include a particular pattern, we will double count those that include two or more of the patterns. We must rely on the principle of inclusion-exclusion. There are $n!$ total permutations. We count the number of permutations that include a particular 2-pattern: there are 6 such patterns. Each can be placed in one of $n-1$ positions, and the remaining $n-2$ elements are permuted in $(n-2)!$ ways, yielding $6(n-1)(n-2)! = 6(n-1)!$ permutations.

Next we remove those permutations which include two 2-patterns, as they have been double-counted. If these are $(1\ n)$ and $(2\ n)$, the only possible patterns are the 3-patterns $(1\ n\ 2)$ and $(2\ n\ 1)$. Similarly, we can have the 3-patterns $(n-1\ 1\ n)$ and $(n\ 1\ n-1)$. For these four patterns, we have $n-2$ positions to place them in the permutation, and $(n-3)!$

ways to permute the remaining elements, for a total of $4(n-2)!$ permutations. Now, if we include the two patterns $(2\ n)$ and $(1\ n-1)$, which do not share any elements in common, we have $(n-2)(n-3)$ positions to place them, $2^2$ ways to orient them, and, lastly, $(n-4)!$ ways to permute the remaining elements, yielding another $4(n-2)!$ permutations.

Finally we count the number of permutations that include all three patterns. There are two possibilities: $(2\ n\ 1\ n{-}1)$ and $(n{-}1\ 1\ n\ 2)$. Again, there are $n{-}3$ positions to place these two patterns, and $(n-4)!$ ways to permute the remaining elements, for a total of $2(n-3)!$. Altogether, we have $|A_{n,n-3}| = n!{-}6(n{-}1)!{+}8(n{-}2)!{-}2(n{-}3)! = (n^3{-}9n^2{+}28n{-}30)(n{-}3)!$.

The expressions $|A_{n,n-2}|$ and $|A_{n,n-3}|$ include a polynomial term and $(n-2c+3)!$ as factors. We have in general,

**Theorem 12.** *Let $c \geq 2$ be a constant. The number of permutations satisfying the $(n-c)$-constraint $|A_{n,n-c}| = B(n)(n-2c+3)!$, where $B(n)$ is a monic polynomial of degree $2c-3$.*

*Proof.* Let $D^j_{n,k}$ represent the number of permutations in $S_n$ that contain (at least) $j$ 2-patterns that do not meet the $k$-constraint. We will give an expression for $D^j_{n,k}$.

First, we must avoid the following 2-patterns:

$$\{(n\ 1), (n\ 2), \ldots, (n\ c-1), (n-1\ 1), (n-1\ 2), \ldots, (n-1\ c-2), \ldots, (n-c+2\ 1)\},$$

along with their reflections. There are a total of $\frac{c(c-1)}{2}$ such patterns (and $c(c-1)$ including the reflections). However, not all 2-patterns can be included in a single permutation. For example, it is not possible to find the patterns $(n\ 2)$ and $(n-1\ 2)$ in the same permutation. In fact, the largest number of 2-patterns to be found in a single permutation is $2c-3$. This can be seen from the following $(2c-2)$-pattern:

$$(c-1\ n\ c-2\ n-1\ c-3\ n-2 \ldots\ 2\ n-c+3\ 1\ n-c+2).$$

Thus, we may write the following expression for $|A_{n,k}|$ :

$$|A_{n,k}| = n! - \sum_{j=1}^{2c-3} (-1)^{j+1} D_{n,k}^{j},$$

by the principle of inclusion-exclusion. Essentially, we count the number of permutations including one 2-pattern. In doing this, we have double counted those permutations that include two 2-patterns, so we must remove these. Now we have twice removed those which have exactly three 2-patterns, so we place them back in, and so on.

Next we compute $D_{n,k}^{j}$, the number of permutations that include $j$ 2-patterns. Note that these patterns may be separate, or may be part of an $\ell$-pattern, where $\ell > 2$. For example, the patterns $(1\ n)$ and $(2\ n-3)$ are separate, as they do not have elements in common. On the other hand, $(1\ n\ 2)$ contains the 2-patterns $(1\ n)$ and $(n\ 2)$ consecutively. Let us say that our $j$ 2-patterns form $w$ separate patterns $p_1, \ldots, p_w$, no two of which have any permutation elements in common.

To see how this works, let us take the case $c = 5$ and $j = 5$. Then, the following set of (separate) patterns can be included in one permutation:

$$\{(3\ n-1\ 2\ n-2), (4\ n), (1\ n-3)\}.$$

The first 4-pattern contains three consecutive 2-patterns, so that indeed there are a total of $j = 5$ 2-patterns. Since there are three seperate "parts", $w = 3$.

Given $j$ and $w$, we compute how many permutations there are with suitable patterns. Let us say we select the $j$ patterns (in $w$ parts) from the $\frac{c(c-1)}{2}$ total patterns in $L_{j,w}$ ways. We may reflect any of the separate patterns, yielding a factor of $2^w$. Next, we note that the patterns contain $w + j$ total permutation elements. This can be seen from the following argument: there are $2j$ elements in $j$ 2-patterns, but all those which are not the first or last element in a part are counted in two distinct 2-patterns, so they must be subtracted. There are $j - w$ such elements, so that $2j - (j - w) = j + w$ gives the number of permutation

elements. Thus, there are $n-w-j$ permutation elements not involved in the patterns, which may be permuted in $(n-w-j)!$ ways. Lastly, there are now $n-j$ positions to place the separate patterns in the permutation, and $w!$ ways to permute them. Thus, we have a total of

$$2^w \binom{n-j}{w} w!(n-w-j)!L_{j,w}$$

permutations, which is equal to $2^w(n-j)!L_{j,w}$. Then,

$$\begin{aligned} D^j_{n,k} = \sum_w 2^w(n-j)!L_{j,w} = (n-j)! \left( \sum_w 2^w L_{j,w} \right) \\ = (n-j)\cdots(n-2c+4)(n-2c+3)! \left( \sum_w 2^w L_{j,w} \right) \\ = E^j(n)(n-2c+3)!, \end{aligned}$$

where $E^j(n)$ is a polynomial with degree $2c-3-j$. Thus,

$$|A_{n,k}| = ((n)\cdots(n-2c+4) - \sum_{j=1}^{2c-3} (-1)^{j+1} E^j(n))(n-2c+3)!.$$

Now the result is immediate. The largest degree of $E^j(n)$ is $2c-4$ when $j=1$, while $n(n-1)\cdots(n-2c+4)$ has degree $2c-3$ and leading coefficient 1, as desired. □

The above theorem also immediately confirms that the capacity of $(n-c)$-constrained codes is 1.

# CHAPTER 4

# Conclusion

## 4.1 Summary of Our Results

In this thesis, we explored coding techniques to improve the reliability and lifetimes of NVM devices. NVMs have become the most popular data storage technology. Furthermore, there is extraordinary demand for increased storage capacities. For these reasons, reliability has become a critical issue in NVMs, motivating our application of coding-theoretic strategies to these technologies.

Two approaches were studied. First, we extended the dynamic thresholds scheme introduced in [ZJB11] to the multi-level cell case. We showed that the advantages of dynamic thresholds are preserved in the MLC case. In particular, error probabilities are reduced when using dynamic thresholds versus fixed thresholds. This was demonstrated through an analysis of small cases and by simulations. We also showed that dynamic thresholds perform nearly as well as the unobtainable optimal thresholding scheme.

We introduced algorithms to allow decoders to compute dynamic thresholds. These algorithms came in two flavors: the "metadata" approach and the "balanced codes" approach. We also commented on issues that must be resolved in order to make dynamic thresholds a practical choice in NVM chips.

The combination of dynamic thresholds with error-correcting codes was explored in depth. We sought code constructions that are tailored to correct exactly those errors which are ex-

perienced when using dynamic thresholds. In the error model where only a limited number of (limited-magnitude) errors may take place, we connected this problem with location correction (when using the metadata approach) and with generalized rank modulation (when using the balanced codes approach.) We also introduced a new construction that corrects any number of limited-magnitude errors.

In the second part of this work, we studied the concept of constrained rank modulation. The purpose of this technique is to resolve the inter-cell coupling issue when using the permutation-based rank modulation scheme. We introduced various constraints that are capable of reducing the effects of inter-cell coupling. In particular, we analyzed the single neighbor $k$-constraint, where adjacent elements of the rank modulation permutation may differ by at most $k$.

For the single neighbor $k$-constraint, we computed bounds and gave an efficient construction which is asymptotically optimal. We studied the overall asymptotics of the scheme. Lastly, we examined two useful special cases, where the constraint parameter $k$ is either a constant or within a constant of the code length $n$.

Our work on dynamic thresholds for MLC NVMs has been presented in preliminary form at the *IEEE Asilomar Conference on Signals, Systems, and Computers* in Nov. 2012 [SGD12]. A longer version of this work has been accepted for publication to *IEEE Transactions on Communications* in 2013 [SGD13]. The results regarding constrained rank modulation have been submitted to the 2013 *IEEE Information Theory Workshop* (ITW) [SD13a]. Finally, another work, "Counting Sequences Obtained From the Synchronization Channel," [SD13b] will be presented at the 2013 *IEEE International Symposium on Information Theory* (ISIT). This work, though not detailed in this thesis, is closely tied to the goal of improving the reliability of NVM devices. The paper studies a certain subclass of the synchronization channel, which allows for the deletion of symbols from (or the insertion of spurious symbols into) a message. Such channels model, for example, storage systems with backup, where small changes to the information stored in one system can be seen as the result of insertions

and deletion errors from the perspective of the backup system.

## 4.2 Future Directions

There are several possibilities available for future work. For dynamic thresholds, it remains to be determined in the general case whether the code constructions based on the metadata implementation are superior to those based on the balanced-codes approach. The metadata-based codes are linear block codes with easily computed rates. However, the balanced code constructions are much more difficult to analyze.

The balanced codes are an interesting subject in their own right. These codes can also be described as error-correcting constant-weight codes or error-correcting multipermutation codes. Such codes have applications beyond dynamic thresholds. Our development in Chapter 2 can serve as a starting point towards a more complete analysis of such codes. In particular, it would be interesting to compute sphere-packing and Singleton-like bounds based on the generalized Kendall tau distance we have given. Similarly, it would be useful to search for balanced code constructions that correct a number of errors that is a fraction of the code length. Computing the capacity of such codes is another direction for future work. It may be possible to derive some of these results by generalizing similar ideas developed for permutation-based codes in [MBZ13].

Another question regarding dynamic thresholds is whether there is a benefit to mixing the two implementations: that is, requiring that codewords are balanced within a certain degree, so that for codewords $\mathbf{x}_1$ and $\mathbf{x}_2$, $\mathbf{k}(\mathbf{x}_1)$ and $\mathbf{k}(\mathbf{x}_2)$ differ in each component only by a limited amount. The extreme cases where we either require $\mathbf{k}(\mathbf{x}_1) = \mathbf{k}(\mathbf{x}_2)$ or allow $\mathbf{k}(\mathbf{x}_1)$ and $\mathbf{k}(\mathbf{x}_2)$ to differ by any amount represent the balanced codes and metadata approaches, respectively.

There are several potential directions for future work regarding constrained rank modulation as well. First, it would be useful to generalize the results we have derived for the single neighbor $k$-constraint to the two neighbor $k$-constraint and the two-dimensional $k$-

constraint cases. The latter is particularly interesting, as it correctly models how cells are actually organized in NVM blocks.

Another important aspect is developing a specific model for inter-cell coupling. This is difficult, since the physical interactions that govern inter-cell coupling are quite complicated. However, such a model would enable us to determine what the values of the parameter $k$ should be in general. Furthermore, we would be able to specify exactly how much performance each constrained rank modulation scheme provides in terms of reducing error rates.

Lastly, we would also like to combine constrained rank modulation with rank modulation codes, such as those developed in [BM10] and [MBZ13]. Of course, as a starting point, we could puncture existing codes by removing all codewords that do not meet a $k$-constraint. However, we would like to develop efficient constructions that produce only $k$-constrained codewords. A further interesting question is to determine how much code rates are affected by imposing various constraints.

# References


[AAKT04] R. Ahlswede, H. Aydinian, L. Khachatrian, and L. Tolhuizen, "On $q$-ary codes correcting all unidirectional errors of a limited magnitude," in *Proc. of the 9th International Workshop on Algebraic and Combinatorial Coding Theory*, Kranevo, Bulgaria, Jun. 2004.

[AK08] S. Augustinovich and S. Kitaev, "On uniquely $k$-determined permutations," *Discrete Mathematics*, vol. 308, no. 9, pp. 1500–1507, Sept. 2008.

[BB11] A. Berman and Y. Birk, "Constrained flash memory programming," in *Proc. of the IEEE International Symposium on Information Theory*, St. Petersburg, Russia, Jul.-Aug. 2011, pp. 2128–2132.

[BC62] R. Bose and S. Chowla, "Theorems in the additive theory of numbers," *Commentarii Mathematici Helvetici*, vol. 37, no. 1, pp. 141–147, Dec. 1962.

[BCMV03] R. Bez, E. Camerlanghi, A. Modelli, and A. Visconti, "Introduction to flash memory," *Proc. of the IEEE*, vol. 91, no. 4, pp. 489–502, Apr. 2003.

[Ber61] J. Berger, "A note on error detection codes for asymmetric channels," *Information and Control*, vol. 4, no. 1, pp. 68–73, Mar. 1961.

[BFP+73] M. Blum, F. Floyd, V. Pratt, R. Rivest, and R. Tarjan, "Time bounds for selection," *Journal of Computer and System Sciences*, vol. 7, no. 4, pp. 448–461, Aug. 1973.

[BM10] A. Barg and A. Mazumdar, "Codes in permutations and error correction for rank modulation," *IEEE Transactions on Information Theory*, vol. 56, no. 7, pp. 3158–3165, Jul. 2010.

[BS85] L. Babai and V. Sos, "Sidon sets in groups and induced subgraphs of cayley graphs," *European Journal of Combinatorics*, vol. 6, no. 2, pp. 101–114, 1985.

[CGOZ99] P. Cappelletti, C. Golla, P. Olivo, and E. Zanoni, *Flash memories*. Boston, MA: Kluwer, 1999.

[CK69] H. Chadwick and L. Kurz, "Rank permutation group codes based on Kendall's correlation statistic," *IEEE Transactions on Information Theory*, vol. 15, no. 2, pp. 306–315, Mar. 1969.

[CL07] Y. Chee and S. Ling, "Constructions for $q$-ary constant-weight codes," *IEEE Transactions on Information Theory*, vol. 53, no. 1, pp. 135–146, Jan. 2007.

[CSBB10] Y. Cassuto, M. Schwartz, V. Bohossian, and J. Bruck, "Codes for asymmetric limited-magnitude errors with application to multi-level flash memories," *IEEE Transactions on Information Theory*, vol. 56, no. 4, pp. 1582–1596, Apr. 2010.

[EB10] N. Elarief and B. Bose, "Optimal, systematic, $q$-ary codes correcting all asymmetric and symmetric errors of limited magnitude," *IEEE Transactions on Information Theory*, vol. 56, no. 3, pp. 979–984, Mar. 2010.

[ELSB11a] E. En Gad, M. Langberg, M. Schwartz, and J. Bruck, "Constant-weight Gray codes for local rank modulation," *IEEE Transactions on Information Theory*, vol. 57, no. 11, pp. 7431–7442, Nov. 2011.

[ELSB11b] ——, "Generalized Gray codes for local rank modulation," in *Proc. of the IEEE International Symposium on Information Theory*, St. Petersburg, Russia, Jul.-Aug. 2011, pp. 874–878.

[FHV99] F. Fu and A. Han Vinck, "On the capacity of generalized write-once memory with state transitions described by an arbitrary directed acyclic graph," *IEEE Transactions on Information Theory*, vol. 45, no. 1, pp. 308–313, Sept. 1999.

[FS84] A. Fiat and A. Shamir, "Generalized 'write-once' memories," *IEEE Transactions on Information Theory*, vol. 30, no. 3, pp. 470–480, Sept. 1984.

[FSM12] F. Farnoud, V. Skachek, and O. Milenkovic, "Rank modulation for translocation error correction," in *Proc. of the IEEE International Symposium on Information Theory*, Cambridge, MA, Jul. 2012, pp. 2988–2992.

[GCC+09] L. Grupp, A. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. Siegel, and J. Wolf, "Characterizing Flash memory: anomalies, observations, and applications," in *Proc. of the IEEE/ACM International Symposium on Microarchitecture*, New York, NY, Dec. 2009, pp. 24–33.

[GDon] R. Gabrys and L. Dolecek, "Bounds and simple constructions of non-binary write-once memory (WOM)-codes for multilevel flash memories," *IEEE Transactions on Information Theory*, submitted for publication.

[JBB07] A. Jiang, V. Bohossian, and J. Bruck, "Floating codes for joint information storage in write asymmetric memories," in *Proc. of the IEEE International Symposium on Information Theory*, Nice, France, Jun. 2007, pp. 1166–1170.

[JLB12] A. Jiang, Y. Li, and J. Bruck, "Bit-fixing codes for multi-level cells," in *Proc. of the IEEE Information Theory Workshop*, Lausanne, Switzerland, Sept. 2012, pp. 252–256.

[JMB09] A. Jiang, R. Mateescu, and J. Bruck, "Rank modulation for Flash memories," *IEEE Transactions on Information Theory*, vol. 55, no. 6, pp. 2659–2673, Jun. 2009.

[JSB10] A. Jiang, M. Schwartz, and J. Bruck, "Correcting charge-constrained errors in the rank-modulation scheme," *IEEE Transactions on Information Theory*, vol. 56, no. 5, pp. 2112–2121, May 2010.

[JZWB11] A. Jiang, H. Zhou, Z. Wang, and J. Bruck, "Patterned cells for phase change memories," in *Proc. of the IEEE International Symposium on Information Theory*, St. Petersburg, Russia, Jul.-Aug. 2011, pp. 2333–2337.

[KBE11] T. Kløve, B. Bose, and N. Elarief, "Systematic, single limited magnitude error correcting codes for flash memories," *IEEE Transactions on Information Theory*, vol. 57, no. 7, pp. 4477–4487, Jul. 2011.

[Ken38] M. Kendall, "A new measure of rank correlation," *Biometrika*, vol. 30, no. 1-2, pp. 81–93, Jun. 1938.

[Knu86] D. Knuth, "Efficient balanced codes," *IEEE Transactions on Information Theory*, vol. 32, no. 1, pp. 51–53, Jan. 1986.

[KYS+10] S. Kayser, E. Yaakobi, P. Siegel, A. Vardy, and J. Wolf, "Multiple-write WOM codes," in *Proc. of the IEEE Allerton Conference on Communications, Control, and Computing*, Monticello, IL, Sept. 2010, pp. 1062–1068.

[MBZ13] A. Mazumdar, A. Barg, and G. Zemor, "Constructions of rank modulation codes," *IEEE Transactions on Information Theory*, vol. 59, no. 2, pp. 1018–1029, Feb. 2013.

[MSV+09] H. Mahdavifar, P. Siegel, A. Vardy, J. Wolf, and E. Yaakobi, "A nearly optimal construction of flash codes," in *Proc. of the IEEE International Symposium on Information Theory*, Seoul, Korea, Jun.-Jul. 2009, pp. 1239–1243.

[PR04] A. Pirovano and A. Redaelli, "Reliability study of phase-change non-volatile memories," *IEEE Transactions on Device and Materials Reliability*, vol. 4, no. 3, pp. 422–427, Sept. 2004.

[RS82] R. Rivest and A. Shamir, "How to reuse a write-once memory," *Information and Control*, vol. 55, no. 1-3, pp. 1–19, Dec. 1982.

[RS96] R. Roth and G. Seroussi, "Location-correcting codes," *IEEE Transactions on Information Theory*, vol. 42, no. 2, pp. 554–565, Mar. 1996.

[Sch12] M. Schwartz, "Quasi-cross lattice tilings with applications to flash memory," *IEEE Transactions on Information Theory*, vol. 58, no. 4, pp. 2397–2405, Apr. 2012.

[SD13a] F. Sala and L. Dolecek, "Constrained rank modulation," *Submitted to the IEEE Information Theory Workshop*, Sept. 2013.

[SD13b] ——, "Counting sequences obtained from the synchronization channel," in *Proc. of the IEEE International Symposium on Information Theory*, Istanbul, Turkey, Jul. 2013.

[SGD12] F. Sala, R. Gabrys, and L. Dolecek, "Dynamic threshold schemes for multi-level non-volatile memories," in *Proc. of the IEEE Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, Nov. 2012, pp. 1245–1249.

[SGD13] ——, "Dynamic threshold schemes for multi-level non-volatile memories," *accepted to IEEE Transactions on Communications*, 2013.

[Shp12] A. Shpilka, "Capacity achieving multiwrite WOM codes," *Available: http://arxiv.org/pdf/1209.1128v1.pdf*, 2012.

[Sta00] R. Stanley, *Enumerative combinatorics*. Cambridge, UK: Cambridge University Press, 2000.

[TB12a] L. Tallini and B. Bose, "On symmetric $L_1$ distance error control codes and elementary symmetric functions," in *Proc. of the IEEE International Symposium on Information Theory*, Cambridge, MA, Jul. 2012, pp. 741–745.

[TB12b] ——, "On symmetric/asymmetric Lee distance error control codes and elementary symmetric functions," in *Proc. of the IEEE International Symposium on Information Theory*, Cambridge, MA, Jul. 2012, pp. 746–750.

[TS10] I. Tamo and M. Schwartz, "Correcting limited-magnitude errors in the rank-modulation scheme," *IEEE Transactions on Information Theory*, vol. 56, no. 6, pp. 2551–2560, Jun. 2010.

[VT65] R. Varshamov and G. Tenengolts, "Codes which correct single asymmetric errors," *Automatika i Telemekhanika*, vol. 26, no. 2, pp. 288–292, 1965.

[WCSW11] J. Wang, T. Courtade, H. Shankar, and R. Wesel, "Soft information for LDPC decoding in flash: mutual-information optimized quantization," in *Proc. of the IEEE Global Telecommunications Conference (GLOBECOM)*, Houston, TX, Dec. 2011, pp. 1–6.

[WISS13] J. Weber, K. Immink, P. Siegel, and T. Swart, "Polarity-balanced codes," in *Proc. of the IEEE Information Theory Applications Workshop*, San Diego, CA, Feb. 2013.

[YGS+12] E. Yaakobi, L. Grupp, P. Siegel, S. Swanson, and J. Wolf, "Characterization and error-correcting codes for TLC flash memories," in *Proc. of the International Conference on Computing, Networking and Communications.*, Maui, HI, Jan.-Feb. 2012.

[YS12] E. Yaakobi and A. Shpilka, "High sum-rate three-write and non-binary WOM codes," in *Proc. of the IEEE International Symposium on Information Theory*, Cambridge, MA, Jul. 2012, pp. 1386–1390.

[ZJ10] Z. Wang and J. Bruck, "Partial rank modulation for flash memories," in *Proc. of the IEEE International Symposium on Information Theory*, Austin, TX, Jun. 2010, pp. 864–868.

[ZJB09] Z. Wang, A. Jiang, and J. Bruck, "On the capacity of bounded rank modulation for flash memories," in *Proc. of the IEEE International Symposium on Information Theory*, Seoul, Korea, Jun.-Jul. 2009, pp. 1234–1238.

[ZJB11] H. Zhou, A. Jiang, and J. Bruck, "Error-correcting schemes with dynamic thresholds in non-volatile memories," in *Proc. of the IEEE International Symposium on Information Theory*, St. Petersburg, Russia, Jul.-Aug. 2011, pp. 2143–2147.

[ZJB12a] ——, "Balanced modulation for nonvolatile memories," *Available: http://arxiv.org/pdf/1209.0744.pdf*, 2012.

[ZJB12b] ——, "Systematic error-correcting codes for rank modulation," in *Proc. of the IEEE International Symposium on Information Theory*, Cambridge, MA, Jul. 2012, pp. 2978–2982.

[ZPJ10] F. Zhang, H. Pfister, and A. Jiang, "LDPC codes for rank modulation in flash memories," in *Proc. of the IEEE International Symposium on Information Theory*, Austin, TX, Jun. 2010, pp. 859–863.