Title
Channel Coding Strategies for Emerging Data Storage Systems

Author
Gabrys, Ryan C.

Publication Date
2014

Peer reviewed|Thesis/dissertation

University of California

Los Angeles

Channel Coding Strategies for Emerging Data Storage Systems

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in Electrical Engineering

by

Ryan Christopher Gabrys

2014

Abstract of the Thesis

Channel Coding Strategies for Emerging Data Storage Systems

by

Ryan Christopher Gabrys
Doctor of Philosophy in Electrical Engineering
University of California, Los Angeles, 2014
Professor Lara Dolecek, Chair

The on-going data revolution demands storage systems that can store very large quantities of data while being fast, reliable and cheap. Emerging storage technologies such as flash and granular media offer improved densities, faster access times, and are more power-efficient than conventional hard disk drives. The primary drawback associated with these new devices is their high error rate, caused by difficulties in programming, voltage drift, and wear-out. Coding methods used in existing storage applications are based on symmetric, Hamming-type metrics. However, when used in new memory devices, these traditional approaches result in costly overprovisioning. In this work, we present advanced coding-theoretic techniques applicable to modern storage devices that exploit the asymmetries in the underlying physical operations for improved performance. In many cases of interest, the results in this thesis represent the state of the art. Taken collectively, our results can help enable all modern, data-intensive technologies that require reliably storing large quantities of data.

In the first part of this thesis, we consider the problem of constructing write-once-memory codes (WOM-codes). WOM-codes reduce the number of times a storage device is erased. Since for many non-volatile memory (NVM) devices (such as Flash) the device lifetime is closely related to the number of erases, WOM-codes have the potential to significantly improve device lifetime. We show that in many cases the new codes represent the state of

the art results for certain alphabets and code lengths.

In the second part of this thesis, we develop new error-correcting codes for Flash memory. Our approach is to first perform some analysis of data taken from a Flash device. In particular, we show that the errors that occur throughout the lifetime of the Flash device follow prominent patterns. We then develop new error-correcting codes specifically attuned to these patterns and show that the proposed codes outperform traditional error-correcting codes.

In the third part of this thesis, we consider codes capable of correcting synchronization errors for rank modulation systems. A rank modulation system stores information by the relative values between the cells in a memory device. Thus, the information that is read by a rank modulation system is in the form of a permutation. Motivated by this setup, we consider potential synchronization errors that could occur within rank modulation systems that are the result of underlying hardware errors. One of the highlights of this part of the thesis is that we develop new codes for the Ulam metric and show that the proposed codes improve upon the state of the art.

In the fourth and final part of the thesis, we consider the construction of grain-error-correcting codes for granular media recording devices. The predominant errors we seek to correct are the result of the difficulties that arise due to the uncertainty of the grain locations when programming onto the granular medium. The errors manifest themselves as smears whereby the information from one bit overwrites the information stored in an adjacent bit. In many cases, we illustrate how our proposed codes improve upon existing codes.

The dissertation of Ryan Christopher Gabrys is approved.

Richard D. Wesel

Mario Gerla

Alan Laub

Lara Dolecek, Committee Chair

University of California, Los Angeles

2014

To Chrissy, Dash, and Peanut.

Table of Contents

Acknowledgements

Stating all the ways that my advisor, Professor Lara Dolecek, has influenced my research is nearly impossible. Her incredible patience, constant encouragement, and high expectations were instrumental in the completion of this dissertation. Lara guided me to the problem of coding for storage. She had the patience and foresight to help me find subjects that were not only relevant in the field, but were subjects that I personally found interesting. I consider myself incredibly fortunate to have worked with Lara for the past four years, and I am proud of my time spent as a member of the Laboratory for Robust Information Systems.

I am grateful for my dissertation committee, comprised of Professor Richard Wesel, Professor Mario Gerla, Professor Alan Laub, and Professor Lara Dolecek for their support throughout this process. Through classwork and informal discussions on emerging technologies, their time and expertise has helped guide me throughout my time at UCLA.

Professor Eitan Yaakobi has been instrumental to my work over the last three years. As a result of the countless hours we have spent on the white boards at UCSD, Eitan has taught me not only how to approach new research problems, but how to communicate my ideas to others. Without Eitan's enthusiasm, energy, and friendship, this thesis would probably be at least a chapter or two shorter.

I consider myself incredibly lucky to have worked with many talented students and post-docs throughout my PhD experience. Fred Sala has always been willing to listen/engage in almost any type of new problem that I happen to come across. Dr. Farzad Farnoud is responsible for my interest in the subject of coding for rankings which is the subject of the fourth chapter of this book. Based on conversations with Yifan Sun, I became interested in linear programming in the second year of my PhD and the use of this technique constitutes the results in Section 5.3 of this thesis.

Finally, I am grateful to my family. My parents have always emphasized education and

have provided their unwavering support in so many ways that it is impossible to describe them here. I would like to give my deepest gratitude for my amazing wife Chrissy. For four years, she witnessed the effects of my excitement, frustrations, and (at times) despair all the while providing her unconditional support and love. Her willingness to, in many cases, put my happiness ahead of her own is not only selfless but inspiring. My time with her, and our little man, provided me with the balance/perspective I needed to complete this work.

Vita

2005 Bachelor of Science, Mathematics and Computer Science
University of Illinois - Champaign, Urbana

2010 Master of Engineering, Electrical and Computer Engineering
University of California - San Diego

Publications

R. Gabrys, E. Yaakobi, F. Farnoud, F. Sala, J. Bruck, and L. Dolecek, "Single-deletion-correcting codes over permutations," to appear in *Proc. IEEE International Symposium on Information Theory (ISIT)*, Honolulu, HI, July 2014.

R. Gabrys, E. Yaakobi, F. Farnoud, and J. Bruck, "Codes correcting erasures and deletions for rank modulation," to appear in *Proc. IEEE International Symposium on Information Theory (ISIT)*, Honolulu, HI, July 2014.

F. Sala, R. Gabrys, and L. Dolecek, "Deletions in multipermutations," to appear in *Proc. IEEE International Symposium on Information Theory (ISIT)*, Honolulu, HI, July 2014.

R. Gabrys and A. Coker, "Set reconciliation in two rounds of communication," to appear in the *International Command and Control Research and Technology Symposium (ICCRTS)*, Alexandria, VA, June 2014.

F. Sala, R. Gabrys and L. Dolecek, "Dynamic threshold schemes for multi-level non-volatile memories," *IEEE Transactions on Communications*, Vol. 61, No. 7, pp. 2624-2634, 2013.

R. Gabrys, E. Yaakobi, and L. Dolecek, "Correcting grain-errors in magnetic media," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, Istanbul, Turkey, July 2013.

F. Sala, R. Gabrys, and L. Dolecek, "Dynamic threshold schemes for multi-Level nonvolatile memories," *Proc. IEEE Asilomar Conference on Signals, Systems and Computers*, Monterey, CA, Nov. 2012.

R. Gabrys, E. Yaakobi and L. Dolecek, "Graded bit error correcting codes with applications to flash memory," *IEEE Transactions on Information Theory*, Vol. 59, No. 4, pp. 2315-2327, 2013.

R. Gabrys, E. Yaakobi, L. Grupp, S. Swanson, and L. Dolecek, "Tackling intracell variability in TLC flash through tensor product codes," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, Boston, MA, July 2012.

R. Gabrys and L. Dolecek, "Coding for the binary asymmetric channel," in *Proc. IEEE International Conference on Computing, Networking and Communications (ICNC)*, Maui, HI, Jan. 2012.

R. Gabrys and L. Dolecek, "Spatially-aware adaptive error correcting codes for flash memory," in *Proc. IEEE Asilomar Conference on Signals, Systems and Computers*, Monterey, CA, Nov. 2011.

R. Gabrys, E. Yaakobi, L. Dolecek, P. Siegel, A. Vardy, and J. K. Wolf, "Non-binary WOM codes for multilevel flash memories," in *Proc. IEEE Information Theory Workshop (ITW)*, Paraty, Brazil, Oct. 2011.

R. Gabrys and L. Dolecek, "Characterizing capacity achieving write once memory codes for multilevel flash memories," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, St. Petersburg, Russia, July - Aug. 2011.

R. Gabrys and J.K. Wolf, "Modifications to the Lagrange interpolation method of decoding Reed Solomon codes," in *Proc. IEEE International Symposium on Communications and Information Technologies (ISCIT)*. Lao China, Oct. 2008.

CHAPTER 1

Introduction

Storage systems have become almost ubiquitous today. Some of the most popular memory technologies include Flash, memristor, Phase Change Memory (PCM), and granular media recording. However, many of these newer technologies have issues that include high error rates along with limited device lifetimes. In this thesis, we present coding techniques aimed at overcoming these limitations.

The architecture and issues with each storage system differs depending on the technology. As a result, in this section, we briefly describe two of the most popular types of storage devices, Flash memory and granular media recording media, and comment on some of the problems/limitations with these technologies.

1.1 Flash Memory

In this section, we first consider some physical properties of Flash memory. Afterwards, we describe existing coding strategies for Flash memory and detail our contributions.

1.1.1 Overview of Flash Memory

Despite potential benefits that include high data transfer rates, low access times, and reduced power consumption, Flash memory has several potential drawbacks related to the effects of device scaling. Two of the biggest drawbacks to Flash include high error rates and

device wear out. In this section, we briefly describe some of the physical properties of Flash and highlight in more detail the limitations of the hardware.

The atomic unit in a Flash memory device is the floating gate cell. The floating gate cell is a type of metal-oxide semiconductor field transistor. The transistor stores information by forming a negatively charged channel between the source and the drain terminals. This channel appears when voltage is applied to the control gate as depicted in Figure 1.1. A quantum process known as the Fowler-Nordhiem tunneling method then causes electrons to flow through the silicon oxide insulation layer between the floating gate and the P-well. For single level cell (SLC) Flash memories each Flash cell can be in one of two states: 1) the programmed state or 2) the erased state. When the number of electrons in the P-well exceeds some threshold, then the cell is in the programmed state. Otherwise, the cell is in the erased state. For cells capable of storing more than a single bit, such as multiple level cell (MLC) Flash and triple level cell (TLC) Flash, information is stored by introducing more thresholds that are used to represent more than one programmed state. [OL08]

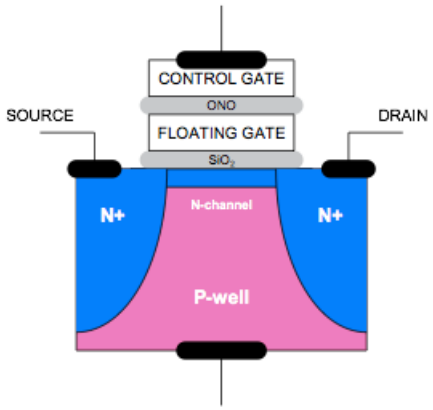

Figure 1.1: Flash Transistor [OL08]

In order to go from a programmed state to the erased state, a high voltage is applied to the silicon substrate while holding the control gate at zero. This causes the electrons that are stored in the P-well to flow through the oxide barrier. Since all the memory cells in a block share a common silicon substrate, it is not possible to erase a single memory cell without

simultaneously erasing an entire block (10^6) of Flash cells. Over time, repeated erases to the same block of cells degrade the oxide insulation layer making the Flash cell unable to store information. [KIN12]

We now provide a brief overview of the architecture of a TLC Flash memory device. A TLC chip is divided into multiple planes. Each plane is divided into a set of blocks and these blocks are further decomposed into pages. A typical configuration for a TLC Flash chip is the following: there are 384 pages within a block and 8 kilobytes (KB) within a page. The eight discrete voltage levels from the cell are represented as a triple-bit word. We refer to the first bit in the word as the most significant bit (MSB), the second bit in the word as the center significant bit (CSB), and the third bit in the word as the least significant bit (LSB). The mapping between voltage levels and triple-bit words is depicted in Table 1.1 [YGSSW12].

Table 1.1: Mapping between Voltage Levels and Triple-bit Words

Voltage Level	Triple-bit Word
0	111
1	110
2	100
3	101
4	001
5	000
6	010
7	011

To reduce programming errors, the information within a TLC Flash cell is programmed three times. More specifically, each Flash cell is programmed once for the most significant bit (MSB), once for the center significant bit (CSB), and once for the least significant bit (LSB). As a result of this programming process, when a Flash cell errs, typically only a single bit of information (one of the MSB, CSB, or LSB) is in error within the cell. This observation will be used to motivate our error-correcting codes in Chapter 3. [YGSSW12]

In the next section, we consider some existing coding techniques that have been proposed to overcome the physical limitations of Flash.

1.1.2 Coding Strategies for Flash Memory

In this section, we consider existing coding strategies for Flash memory and describe our contributions. We begin this section by reviewing work previously performed on write-once-memory systems. Afterwards, we review existing work on the subject of asymmetric error-correction coding with applications to Flash memory. Finally, we provide some background on coding for rank modulation systems. For each of these problems, we comment on how the work in this thesis expands upon known results.

1) **Codes for Write-Once-Memory Systems:** Codes for WOM systems were first introduced by Rivest and Shamir three decades ago [RS82]. In [RS82] binary WOM-codes with the property that once a bit became a 1, it could not transition back to a 0 in subsequent writes were studied. It was demonstrated in [RS82] that WOMs can be rewritten to a surprising degree. In [HEE85, WWZK84], the capacity of such binary WOM-codes was shown to be $\log_2(t+1)$, where t is the number of writes, when the encoder knows the previous state of the memory irrespective of whether the decoder knows the previous state. In [FV99], this result was extended to a WOM-code whose state transitions are representable by a directed acyclic graph. In [JBB07] the model was further extended to include joint storage of multiple variables.

Since the pioneering work of Rivest and Shamir [RS82], the focus has been on designing binary WOM-codes with good properties, as in [CGM86, GOD87] and more recently, in [SHP12a, YKSVW12], and [YS12]. Despite the fact that Fiat and Shamir studied non-binary WOM-codes more than 20 years ago [FS84], until fairly recently there was not much progress on the non-binary set-up.

The design of non-binary WOM-codes is now receiving substantial attention. In [FLM08], codes designed for expected performance were studied. An expression for the capacity region for a fixed number of generations and levels was provided in [FV99]. In [SE13, CY12], and

[KUR11] the design of such codes using lattices was investigated. The focus of [CY12] was on short WOM-codes, and the work presented in [BAR12] investigated a related problem of making short WOM-codes decodable. In [HLA11], error-correcting codes were used to design non-binary WOM-codes. In [YS12], many of the techniques used to construct high-rate binary codes were adapted to construct non-binary WOM codes. Recently, a family of capacity-achieving binary and non-binary WOM-codes was presented in [SHP12b].

In Chapter 2, we construct high-rate non-binary WOM-codes. The non-binary code constructions rely on mappings between binary and non-binary codebooks. Thus, the existence of high-rate codes with short code lengths for the binary WOM implies the existence of high-rate codes with short code lengths for the non-binary WOM. For many code lengths, our constructions produce the best known codes. Since most MLC Flash devices store between 2 and 4 bits of information [Li08], we focus on code constructions capable of producing codes over alphabets of size $4, 8$, and 16.

2) **Asymmetric Error-Correction Codes for Flash Memory:** Recently, the subject of error-correction coding for Flash memory has received significant attention. In [SDRZ06], trellis coded modulation techniques were applied to Flash memory. In [MK09], the use of LDPC codes was investigated, and in [WSW11] it was found that using soft information from multiple reads in the LDPC decoder lowered the error rate. In [HLA11], algebraic error-correction codes were used for rewriting as well as for correcting errors. In [CSBB10, DOL10, EB10, KBE11], codes that correct limited magnitude asymmetric errors were constructed. In [YSVW11], this model was extended to correct graded error patterns. In [SCH11], constructions were given for single error-correcting codes that can correct limited magnitude errors in 2 directions. In [ZJB11], a different error model was considered where the likelihood of an error occurring was directly related to the value of the cell being programmed. The problem was to construct codes that maximized the size of a codebook given some fixed tolerable error probability.

The error model in Chapter 3 is motivated by data collected from a TLC Flash device.

As mentioned previously, if the information from each Flash cell is interpreted as a triple-bit word, then the errors (referred to as graded bit-errors) largely but not exclusively cause only a single bit in each word to change. From this observation, we suggest the use of a class of codes derived from tensor product codes [WOL65] in the context of Flash memory. We refer to this class of codes as **graded bit-error-correcting codes**. The contribution of this part of the thesis is to generalize the result of [YSVW11] to produce code constructions that correct errors that mostly have only a small number of bits in error for each cell-error. In fact, some of the proposed codes indeed end up having the same algebraic structure as generalized tensor product codes (cf. [IF81]). We show that for certain parameters of the constituent codes, such constructions can correct graded bit-errors.

Tensor product codes were first introduced in [WOL65] and were generalized to produce efficient binary codes in [IF81]. In [WOL06], these constructions were revisited and an efficient method of encoding was provided. More recently, tensor product codes were used in the context of magnetic recording [CS06a, CS06b]. In a concatenated coding scheme, the use of a tensor product parity code as the inner code was shown to offer the performance advantages of a short length parity code but without the associated rate penalty. In [AM10], tensor product codes were used in conjunction with soft iterative decoding methods to manage the size of the syndrome table. In this part of the thesis, a new type of generalized tensor product code, the graded bit-error-correcting code, is developed. These codes are demonstrated to correct the errors that occur within a TLC Flash device. In particular, generalized tensor product codes are shown to delay the onset of errors longer than conventional coding schemes. Delaying the onset of errors is significant since the device can potentially be used for a longer period of time.

In contrast to conventional symmetric error-correcting codes (e.g., Chapters 5 and 8 in [ROT06]) that in general correct symbol errors of arbitrary magnitudes, the error correction capability of the proposed graded bit-error-correcting codes is developed only in terms of certain error patterns. This restriction offers performance advantage over symmetric error-

correcting codes for applications, such as Flash, where the errors occur in an asymmetric fashion.

3) **Synchronization Codes for Rank Modulation:** Recall from the previous subsection that Flash memories are comprised of blocks of cells, which can store binary values or can have multiple levels and thus store more than a bit in a cell. For example, a typical block of cells in a Flash memory device contains about 10^6 cells and the number of levels per cell can vary between 2 to 16. One of the main challenges in Flash memories is to exactly program each cell to its level. In order to overcome this difficulty, ***rank modulation codes*** were proposed and studied in [JMSB09]. In this setup, the information is carried by the relative values between the cells rather than by their absolute levels. Thus, every group of cells induces a permutation, which is derived by the ranking of the level of each cell in the group. Shortly after the work in [JMSB09], several works explored codes which correct errors in permutations specifically for the rank modulation scheme; see e.g. [BM10, JSB10, ZJB12]. These works include different metrics such as Kendall's τ, Ulam, and Hamming distances. However, none of the recent works explored the setup where elements in the permutations can be deleted or erased. The goal of this part of the thesis is to establish the foundations and present results for these faulty mechanisms.

The paradigms explored in Chapter 4 are derived from a hardware implementation of the modulation process in rank modulation codes. Assume that a permutation π is stored in the Flash memory cells. While reading π, an error in the hardware may occur where the information from a cell may be completely corrupted. Rather than return the information read from the corrupted cell, we assume the readback mechanism omits the corrupted cell from the ranking. Notice that in this case, a cell was deliberately omitted from the permutation read. We also consider the case where a cell is accidentally omitted from the permutation read. This type of error may occur, for instance, if the readback mechanism skips over a cell when it is scanning a large block of cells. In the case where a cell is accidentally skipped over, the information read back will be in the form of a permutation where the values of

other symbols may be affected as well.

We will consider four different models, which correspond to 1) ***erasure/deletion***: whether the location of the lost cell is known, i.e. an erasure or deletion, and 2) ***stability***: whether the other symbols do or do not change their values as a result of the erasure/deletion. For example, assume that the stored permutation is $(5, 3, 2, 4, 1)$ and the third symbol 2 was either erased or deleted. For the case of stable erasure, the read information is $(5, 3, ?, 4, 1)$; for unstable erasure, the read information is $(4, 2, ?, 3, 1)$; for stable deletion, the read information is $(5, 3, 4, 1)$; and lastly, for unstable deletion, the read information is $(4, 2, 3, 1)$.

Our main contribution in each model of erasures and deletions is to find a known distance metric for permutations that will provide codes in the corresponding model. In particular, we show that codes based upon the Hamming distance can be used to correct stable erasures. We also show that the models of unstable erasures and stable deletions are equivalent and codes in the Ulam distance can be used in these setups.

To the best of our knowledge, the research on codes combatting the proposed models is very limited. We could only specify the work by Levenshtein [LEV91] which falls under the stable deletions model and the follow up works in [TEN84, VT65].

1.2 Granular Media Recording Medium

In this section, we proceed in a manner similar to Section 1.1. We first consider some physical properties of granular media. Afterwards, we consider existing coding strategies for granular media and detail our contributions.

1.2.1 Overview of Granular Media

The atomic unit in magnetic media is a grain. Grains can be magnetized to represent two states so that each grain is capable of storing a single bit of information. In conventional media, the magnetic recording layer is a comprised of a collection of grains which behave as

independent magnetic elements. Under this setup, each bit of information recorded requires a collection of random grains. In an effort to increase the storage capacity of these devices, the use of patterned media has been proposed [WNP97].

Patterned media is designed with well-defined magnetic islands. The write-process encodes information into the positions of the magnetic islands so that each island is capable of storing a single bit of information. One of the challenges to this approach is to create an array of islands with consistent magnetic properties to achieve the desired storage density of one bit of information per grain. A bit patterned media recording device, along with some of the concepts described in these first two paragraphs, is depicted in Figure 1.2 below [HIT14].

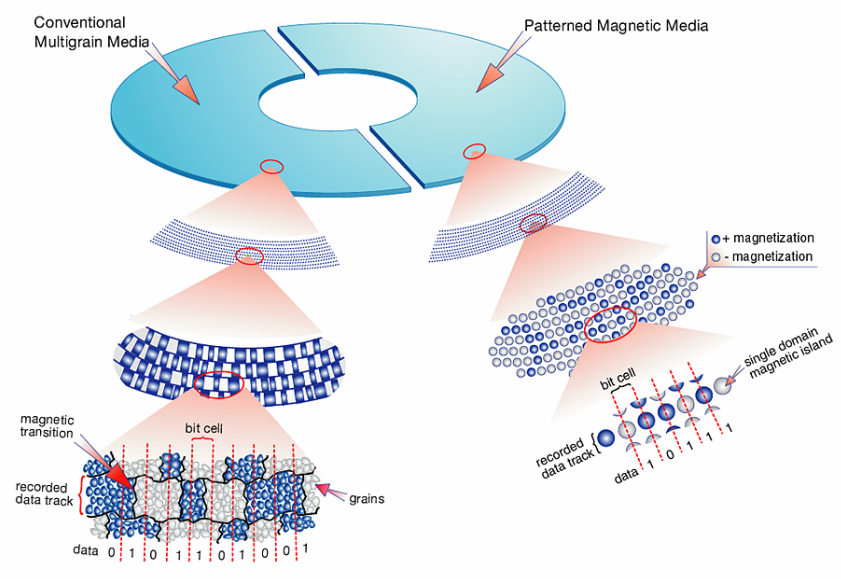

Figure 1.2: Bit Patterned Media [HIT14]

The problem with designing a granular medium with small enough bit islands is that the effects of the random positions of the grains become pronounced. In particular, in [WWKM09] a one-dimensional channel model was studied that illustrated the effects of having grains with randomly selected lengths of 1, 2, or 3 bits. When grains span more than a single bit cell, the polarity of a grain is set by the last bit written into it. The errors manifest themselves as overwrites (or ***smears***) where the last bit in the grain overwrites the preceding bit in the grain. In this part of the thesis, the focus is on grains of length

one or two bits. A ***grain-error*** is an error where the information from one bit overwrites the information stored in the preceding bit in the grain. Without loss of generality, and as in [MBK11], our model assumes that the first bit smears the following adjacent bit in the grain.

1.2.2 Coding Strategies for Granular Media

In [SR11], combinatorial bounds and code constructions were presented for granular media. In [ISW11], the authors studied a related model from an information-theoretic perspective. In [MBK11], a combinatorial channel model was introduced and coding methods were proposed for a one-dimensional granular magnetic medium. In [MBK11], the focus was on binary alphabets and the types of errors studied in [MBK11] will be referred in this thesis as ***non-overlapping grain-errors***. In [SR11], the model was generalized to include non-binary alphabets as well as ***overlapping grain-errors***. Overlapping grain-errors permit the occurrence of two errors in consecutive positions whereas non-overlapping grain-errors cannot be adjacent. Note that there is no distinction between a non-overlapping single grain-error and an overlapping single grain-error. In this thesis, we restrict our attention only to the overlapping grain-error model. We say that a code is a ***t-grain-error-correcting code*** if it can correct up to t overlapping grain-errors. In both [MBK11] and [SR11], bounds and constructions were given. Recently, in [KZ13] some of the techniques from [KK12] were adopted to obtain improved upper bounds on the maximum cardinalities of non-overlapping grain-error codes.

The main contribution of this part of the thesis is to construct codes that correct grain-errors. We show that the class of group codes from [MR79] is a special case of our general code construction. In addition, and similar to [KZ13], we provide non-asymptotic upper bounds on the cardinalities of t-grain-error-correcting codes, with an explicit expression for the cases where $t = 1, 2, 3$. We show that in many cases our bounds and constructions improve upon the state of the art results from [MBK11] and [SR11].

CHAPTER 2

Constructions of Non-Binary WOM-Codes for Multilevel Flash Memories

2.1 Introduction

In this chapter, we present our constructions for non-binary WOM-codes for multilevel Flash memories. In this first section, we begin by describing the notation that will be used throughout the chapter. In Section 2.2, we begin with some simple, yet rate efficient codes. In Section 2.3, we present a construction for a two-write WOM-code over an alphabet of size four. In Section 2.4, we derive bounds and consider constructions for codes that encode a fixed amount of information at every write.

We assume in this work that the information within a multilevel Flash cell can be represented using the q elements from $\mathbb{Z}_q$, where $\mathbb{Z}_q$ represents the ring modulo q. Under this setup, q is the number of levels in a Flash memory device. The initial state of each cell is zero. We assume that while it is possible to increase a cell level, it is not possible to decrease its level. The information from a collection of n q-ary Flash cells is represented as an element from $\mathbb{Z}_q^n$ and we refer to an element from $\mathbb{Z}_q^n$ as a ***cell-state vector***. For two cell-state vectors $\boldsymbol{x} = (x_1, \ldots, x_n), \boldsymbol{y} = (y_1, \ldots, y_n) \in \mathbb{Z}_q^n$, we denote $\boldsymbol{x} \geq \boldsymbol{y}$ if for all $1 \leq i \leq n$, $x_i \geq y_i$.

We follow the definition of WOM-codes in [YKSVW12] and extend it to the non-binary setup. For a WOM-code $\mathcal{C}_q$, we refer to the set of codewords on the j^{th} write as the j^{th}

generation codewords. In the following, let $\boldsymbol{c}_j$ denote the cell-state vector at generation j. In the definition below, we assume that the decoder for the WOM-code is aware of the generation index j.

Definition 1. *For positive integers $n, t, M_1, \ldots, M_t$, an $[n, t; M_1, \ldots, M_t]_q$* ***t-write WOM-code*** *$\mathcal{C}_q$ is a coding scheme on n q-ary cells. The code $\mathcal{C}_q$ is specified by t pairs of encoding and decoding maps $\mathcal{E}_{q,j}$ and $\mathcal{D}_{q,j}$, for $1 \leq j \leq t$, satisfying the following properties:*

1. $\mathcal{E}_{q,1} : \mathbb{Z}_{M_1} \to \mathbb{Z}_q^n$,

2. *For $2 \leq j \leq t$,*

$$\mathcal{E}_{q,j} : \mathbb{Z}_{M_j} \times Im(\mathcal{E}_{q,j-1}) \to \mathbb{Z}_q^n,$$

   where $Im(\mathcal{E}_{q,j-1})$ denotes the image of the map $\mathcal{E}_{q,j-1}$, and for all $(m_j, \boldsymbol{c}_{j-1}) \in \mathbb{Z}_{M_j} \times Im(\mathcal{E}_{q,j-1})$,

$$\mathcal{E}_{q,j}(m_j, \boldsymbol{c}_{j-1}) \geq \boldsymbol{c}_{j-1}.$$

3. *For $1 \leq j \leq t$,*

$$\mathcal{D}_{q,j} : Im(\mathcal{E}_{q,j}) \to \mathbb{Z}_{M_j},$$

   so that $\mathcal{D}_{q,1}(\mathcal{E}_{q,1}(m_1)) = m_1$ for all $m_1 \in \mathbb{Z}_{M_1}$, and for $2 \leq j \leq t$, $\mathcal{D}_{q,j}(\mathcal{E}_{q,j}(m_j, \boldsymbol{c}_{j-1})) = m_j$ for all $(m_j, \boldsymbol{c}_{j-1}) \in \mathbb{Z}_{M_j} \times Im(\mathcal{E}_{q,j-1})$.

Remark 1. *For notational convenience, we may refer to the map $\mathcal{E}_{q,1}$ as $\mathcal{E}_{q,1} : \mathbb{Z}_{M_1} \times (0, 0, \ldots, 0) \to \mathbb{Z}_q^n$, where the second argument is ignored since the cell-state vector is always the all-zeros vector for the first write. For shorthand, on the first write, the current cell-state vector $\boldsymbol{c}_0$ is assumed to be the all-zeros vector.*

Notice that in the above definition our messages are represented as integers between 0 and $M_j - 1$ where $1 \leq j \leq t$ for a t-write WOM-code. For shorthand, we refer to the second property in the definition, which requires that cells never decrease their values, as the ***WOM-constraint***.

Definition 2. *The **rate** of a t-write WOM-code $\mathcal{C}_q$ on the j^{th} write, $1 \leq j \leq t$, is defined to be $\mathcal{R}_j(\mathcal{C}_q) = \frac{\log_2 M_j}{n}$, and the **sum-rate** of the WOM-code $\mathcal{C}_q$ is*

$$\mathcal{R}_{sum}(\mathcal{C}_q) = \sum_{j=1}^{t} \mathcal{R}_j(\mathcal{C}_q).$$

Unless stated otherwise, the log function is base two. If the rates are the same at each write for a t-write WOM-code $\mathcal{C}_q$, then $\mathcal{C}_q$ is said to be a ***fixed-rate WOM-code***.

The maximum sum-rate possible for a WOM-code $\mathcal{C}_q$ will be referred to as the ***capacity***. The maximum sum-rate possible for a fixed-rate WOM-code $\mathcal{C}_q$ will be referred to as the ***fixed-rate capacity***. We have the following result from [FV99].

Lemma 1. *(cf. [FV99]) The capacity for a q-ary t-write WOM-code is* $\log_2 \begin{pmatrix} q+t-1 \\ q-1 \end{pmatrix}$.

2.2 Elementary Constructions

We first present two elementary ways to construct non-binary WOM-codes. The basic idea here is to use mappings between WOM-codes defined over small alphabets and WOM-codes for larger alphabets. For certain code lengths, the codes detailed below represent the state of the art.

2.2.1 Expansion Construction

The main idea behind the ***Expansion Construction*** is to independently encode k q-ary WOM-codes and, using a map, represent these WOM-codes over an alphabet of size q^k. This method of encoding information using multiple binary ($q = 2$) codebooks will be improved upon in Section 2.3 where it is shown that higher sum-rates are achievable for a resultant 4-ary WOM-code if we no longer restrict ourselves to independent binary codebooks (at the additional cost of possibly increasing the length of the codebook).

Let q, k be two positive integers. The map $\phi_{q,k} : \mathbb{Z}_q^k \to \mathbb{Z}_{q^k}$ is defined such that for $\boldsymbol{x} = (x_1, \ldots, x_k) \in \mathbb{Z}_q^k$,

$$\phi_{q,k}(\boldsymbol{x}) = \Sigma_{s=1}^{k} q^{k-s} x_s.$$

Here it is clear that $\boldsymbol{x}$ represents the q-ary expansion of $\phi_{q,k}(\boldsymbol{x})$. Since this map is one-to-one and onto, its inverse map $\phi_{q,k}^{-1}$ exists. For a symbol $c \in \mathbb{Z}_{q^k}$, let $\phi_{q,k}^{-1}(c)_s$ denote the s^{th} element of the q-ary expansion of c.

Lemma 2 below constructs a 2-write q^k-ary WOM-code of length n by considering the q-ary expansion for each of the n symbols in the memory. For s, $1 \leq s \leq k$, the s^{th} q-ary digit from all symbols are collected into a vector from $\mathbb{Z}_q^n$, and then this vector from $\mathbb{Z}_q^n$ is encoded according to a q-ary WOM-code of length n. The Expansion Construction is presented in a slightly informal manner for the two-write case in the lemma below. A more general version of the construction is given below.

Lemma 2. *If there exists an $[n, 2; M_1, M_2]_q$ 2-write WOM-code $\mathcal{C}_q$, then there exists an $[n, 2; M_1^k, M_2^k]_{q^k}$ 2-write WOM-code $\mathcal{C}_{q^k}$.*

Proof. We prove the result by describing the encoding and decoding procedure. For the first write, we choose k first generation codewords, say $\boldsymbol{c}_1^{(1)}, \boldsymbol{c}_1^{(2)}, \ldots, \boldsymbol{c}_1^{(k)} \in \mathbb{Z}_q^n$ from the codebook $\mathcal{C}_q$. For $1 \leq j \leq k$, let $\boldsymbol{c}_1^{(j)} = (c_{1,1}^{(j)}, c_{1,2}^{(j)}, \ldots, c_{1,n}^{(j)})$ so that $c_{1,i}^{(j)}$ refers to the i^{th} bit in $\boldsymbol{c}_1^{(j)}$. We then use the map $\phi_{q,k}$ to produce our first generation codeword $\boldsymbol{c}_1 = (c_{1,1}, c_{1,2}, \ldots, c_{1,n}) \in \mathcal{C}_{q^k}$ where for $1 \leq i \leq n$, we have $c_{1,i} = \phi_{q,k}(c_{1,i}^{(1)}, \ldots, c_{1,i}^{(k)})$. Since the map $\phi_{q,k}$ is invertible it is straightforward to recover the codewords $\boldsymbol{c}_1^{(1)}, \boldsymbol{c}_1^{(2)}, \ldots, \boldsymbol{c}_1^{(k)}$ from $\boldsymbol{c}_1$. Since there are M_1 first generation messages in $\mathcal{C}_q$ and we choose k of them (by picking k codewords in $\mathcal{C}_q$) there are M_1^k total first generation messages in $\mathcal{C}_{q^k}$.

For the second write, we first recover the k first generation codewords $\boldsymbol{c}_1^{(1)}, \boldsymbol{c}_1^{(2)}, \ldots, \boldsymbol{c}_1^{(k)} \in \mathbb{Z}_q^n$ used to encode $\boldsymbol{c}_1$. Then we choose k second generation codewords from $\mathcal{C}_q$. More specifically, we choose a second codeword $\boldsymbol{c}_2^{(1)}$ provided the cell-state vector $\boldsymbol{c}_1^{(1)}$ so that $\boldsymbol{c}_2^{(1)} \geq \boldsymbol{c}_1^{(1)}$ (this property follows since $\mathcal{C}_q$ is a two-write WOM-code). Similarly we choose

a second generation codeword $\boldsymbol{c}_2^{(2)}$ provided the cell-state vector $\boldsymbol{c}_1^{(2)}$; we choose a second generation codeword $\boldsymbol{c}_2^{(3)}$ provided the cell-state vector $\boldsymbol{c}_1^{(3)}$ and so on. Thus, we have a total of k second generation codewords $\boldsymbol{c}_2^{(1)}, \ldots, \boldsymbol{c}_2^{(k)}$. For $1 \leq j \leq k$, let $\boldsymbol{c}_2^{(j)} = (c_{2,1}^{(j)}, c_{2,2}^{(j)}, \ldots, c_{2,n}^{(j)})$ so that $c_{2,i}^{(j)}$ refers to the i^{th} bit of $\boldsymbol{c}_2^{(j)}$. We now use the map $\phi_{q,k}$ to produce the second generation codeword $\boldsymbol{c}_2 = (c_{2,1}, c_{2,2}, \ldots, c_{2,n}) \in \mathcal{C}_{q^k}$ so that for $1 \leq i \leq n$, $c_{2,i} = \phi_{q,k}(c_{2,i}^{(1)}, \ldots, c_{2,i}^{(k)})$. Since $\mathcal{C}_q$ is a two-write WOM-code, for $1 \leq j \leq k$, we have $\boldsymbol{c}_2^{(j)} \geq \boldsymbol{c}_1^{(j)}$ and so $\boldsymbol{c}_2 \geq \boldsymbol{c}_1$ so that the WOM-constraint is satisfied. Since the map $\phi_{q,k}$ is invertible it is straightforward to recover $\boldsymbol{c}_2^{(1)}, \ldots, \boldsymbol{c}_2^{(k)}$ provided $\boldsymbol{c}_2$. Since there are M_2 second generation messages in $\mathcal{C}_q$ and we choose k of them (by picking k codewords in $\mathcal{C}_q$) there are M_2^k total second generation messages in $\mathcal{C}_{q^k}$. □

A more general (and more formal) presentation of the Expansion Construction is stated below.

Lemma 3. *If there exists an $[n, t; M_1, \ldots, M_t]_q$ t-write WOM-code $\mathcal{C}_q$, then there exists an $[n, t; M_1^k, \ldots, M_t^k]_{q^k}$ t-write WOM-code.*

Proof. Assume that t encoding and decoding maps of the $[n, t; M_1, \ldots, M_t]_q$ t-write WOM-code $\mathcal{C}_q$ are denoted by $\mathcal{E}_{q,j}, \mathcal{D}_{q,j}$, for j where $1 \leq j \leq t$. We construct an $[n, t; M_1^k, \ldots, M_t^k]_{q^k}$ t-write WOM-code which we denote by $\mathcal{C}_{q^k}$ with encoding and decoding maps $\mathcal{E}_{q^k,j}, \mathcal{D}_{q^k,j}$, for j where $1 \leq j \leq t$, as follows.

On the j^{th} write ($1 \leq j \leq t$) the input to the map $\mathcal{E}_{q^k,j}$ is $(m_j, \boldsymbol{c}_{j-1})$ where $m_j \in \mathbb{Z}_{M_j^k}$ and $\boldsymbol{c}_{j-1} = (c_1, \ldots, c_n) \in \mathbb{Z}_{q^k}^n$. Recall that

$$\phi_{M_j,k}^{-1}(m_j) = (m_{j,1}, m_{j,2}, \ldots, m_{j,k}) \tag{2.1}$$

is the M_j-ary expansion of m_j. In the following, for all s where $1 \leq s \leq k$, $\phi_{q,k}^{-1}(c)_s$ denotes the s^{th} element of the q-ary expansion of c, where $c \in \mathbb{Z}_{q^k}$. Notice that $\phi_{M_j,k}^{-1}(m_j)_s = m_{j,s}$.

We construct vectors

$$\boldsymbol{z}^{(s)} = (\phi_{q,k}^{-1}(c_1)_s, \ldots, \phi_{q,k}^{-1}(c_n)_s), \tag{2.2}$$

where $\boldsymbol{z}^{(s)} \in \mathbb{Z}_q^n$. We have

$$\boldsymbol{u}^{(s)} = \mathcal{E}_{q,j}(m_{j,s}, \boldsymbol{z}^{(s)}), \tag{2.3}$$

where $\boldsymbol{u}^{(s)} = (u_1^{(s)}, \ldots, u_n^{(s)}) \in \mathbb{Z}_q^n$ is the output of the encoder $\mathcal{E}_{q,j}$ applied to $\boldsymbol{z}^{(s)}$.

Then, for all i where $1 \leq i \leq n$, let $\boldsymbol{v}^{(i)} = (u_i^{(1)}, \ldots, u_i^{(k)})$. In this case, $\boldsymbol{v}^{(i)}$ is comprised of the i^{th} q-ary symbol from each of the $\boldsymbol{u}^{(s)}$ vectors (where $1 \leq s \leq k$). Define the output of the encoding map to be

$$\mathcal{E}_{q^k,j}(m_j, \boldsymbol{c}_{j-1}) = \left(\phi_{q,k}(\boldsymbol{v}^{(1)}), \ldots, \phi_{q,k}(\boldsymbol{v}^{(n)})\right). \tag{2.4}$$

Note that since $\mathcal{C}_q$ is a WOM-code, then for all s where $1 \leq s \leq k$, $\boldsymbol{u}^{(s)} \geq \boldsymbol{z}^{(s)}$ and hence $\mathcal{E}_{q^k,j}(m_j, \boldsymbol{c}_{j-1}) \geq \boldsymbol{c}_{j-1}$ since the encoder $\mathcal{E}_{q,j}$ is applied component-wise. Therefore, the WOM-constraint is satisfied.

Similarly, for $1 \leq j \leq t$, we define the decoding map $\mathcal{D}_{q^k,j} : \mathbb{Z}_{q^k}^n \to \mathbb{Z}_{M_j^k}$ as follows. Let $\boldsymbol{c}_j = (c'_1, \ldots, c'_n) \in \mathbb{Z}_{q^k}^n$ be the cell-state vector after the j^{th} write. For all s where $1 \leq s \leq k$, let

$$\hat{m}_{j,s} = \mathcal{D}_{q,j}(\phi_{q,k}^{-1}(c'_1)_s, \ldots, \phi_{q,k}^{-1}(c'_n)_s), \tag{2.5}$$

and

$$\mathcal{D}_{q^k,j}(\boldsymbol{c}_j) = \phi_{M_j,k}(\hat{m}_{j,1}, \ldots, \hat{m}_{j,k}). \tag{2.6}$$

From (2.3) and (2.4), it can be seen by substitution that the input to $\mathcal{D}_{q,j}$ in (2.5) is $\boldsymbol{u}^{(s)}$.

Thus, we have $\hat{m}_{j,s} = \mathcal{D}_{q,j}(\mathcal{E}_{q,j}(m_{j,s}, \boldsymbol{z}^{(s)})) = m_{j,s}$ and so $\mathcal{D}_{q^k,j}(\boldsymbol{c}_j) = \phi_{M_j,k}(\hat{m}_{j,1}, \ldots, \hat{m}_{j,k}) = m_j$, as desired. □

For shorthand, we will refer to the construction presented in the proof of Lemma 3 as the Expansion Construction. The code $\mathcal{C}_q$ in the statement of the lemma will be referred to as the base WOM-code. The encoding procedure for a code created with the Expansion Construction is illustrated in the following example.

Example 1. *Let $\mathcal{C}_2$ be the well-known Rivest-Shamir $[3, 2; 4, 4]_2$ two-write WOM-code [RS82] illustrated in Table 1 that can encode 4 messages in the first generation and 4 messages in the second generation. Recall from [RS82], that if the same message is encoded twice then, on the second generation, the state of the memory remains the same.*

Table 2.1: Rivest-Shamir two-write WOM-code [RS82]

Message	Generation 1	Generation 2
0	000	111
1	001	110
2	010	101
3	100	011

We use $\mathcal{C}_2$ as the base code in order to construct a $[3, 2; 4^3, 4^3]_{2^3}$ two-write WOM-code of length 3 over $\mathbb{Z}_8$. As in the proof of Lemma 2, we proceed by first choosing three first generation codewords $\boldsymbol{c}_1^{(1)}, \boldsymbol{c}_1^{(2)}, \boldsymbol{c}_1^{(3)}$ from $\mathcal{C}_2$. Suppose $\boldsymbol{c}_1^{(1)} = (0, 1, 0), \boldsymbol{c}_1^{(2)} = (0, 0, 1)$, and $\boldsymbol{c}_1^{(3)} = (0, 1, 0)$. Then, applying the map $\phi_{2,3}$ as described in Lemma 2 to these codewords produces the codeword $\boldsymbol{c}_1 = (\phi_{2,3}(0, 0, 0), \phi_{2,3}(1, 0, 1), \phi_{2,3}(0, 1, 0)) = (0, 5, 2)$.

For the second generation, suppose we choose three second generation codewords $\boldsymbol{c}_2^{(1)}, \boldsymbol{c}_2^{(2)}, \boldsymbol{c}_2^{(3)}$ from $\mathcal{C}_2$. The codeword $\boldsymbol{c}_2^{(1)}$ is chosen provided that the cell-state vector is $\boldsymbol{c}_1^{(1)}$, and similar for $\boldsymbol{c}_2^{(2)}$, $\boldsymbol{c}_2^{(3)}$. Suppose the second codewords chosen are $\boldsymbol{c}_2^{(1)} = (1, 1, 1)$, $\boldsymbol{c}_2^{(2)} = (0, 1, 1)$, and $\boldsymbol{c}_2^{(3)} = (1, 1, 0)$. Since $\mathcal{C}_2$ is a two-write WOM-code we have that $\boldsymbol{c}_2^{(1)} \geq \boldsymbol{c}_1^{(1)}$, $\boldsymbol{c}_2^{(2)} \geq \boldsymbol{c}_1^{(2)}$, and $\boldsymbol{c}_2^{(3)} \geq \boldsymbol{c}_1^{(3)}$. Similar to before, we apply the map $\phi_{2,3}$ to produce the codeword $\boldsymbol{c}_2 = (5, 7, 6)$. This process is summarized in the table below.

Table 2.2: Example of Encoding with Expansion Construction

Write number	Encoding by the base-code $\mathcal{C}_2$	Encoded values in the 8-ary cell
1	(000,101,010)	(0,5,2)
2	(101,111,110)	(5,7,6)

It is easy to deduce the following.

Corollary 1. *If $\mathcal{R}_A$ is the sum-rate of the base WOM-code C_q in Lemma 3, then the sum-rate of the WOM-code $\mathcal{C}_{q^k}$ created according to Lemma 3 is $k \cdot \mathcal{R}_A$.*

The construction from Lemma 3 is capable of producing codes over an alphabet of size 2^k given k identical binary codebooks. In particular, using the binary codes from [YKSVW12], the Expansion Construction from Lemma 3 can produce two-write codes of lengths $16, 23$, and 33 over $\mathbb{Z}_4$ with rates $2.9132, 2.9264$, and 2.9856 respectively. These rates are the best known for this alphabet size and code lengths. We briefly note that the results using the construction ideas from [YS12] for non-binary codes are not reported for $q = 4$ and it is not clear that the techniques from [YS12] produce high-rate codes when $q < 8$ (where q represents the alphabet size).

The following corollary offers a generalization of Lemma 3 where there are two distinct base codes (whereas Lemma 3 had a single base code). Corollary 2 will be used in Section 2.3.

Corollary 2. *If there exists an $[n, t; M_1, \ldots, M_t]_q$ t-write WOM-code $\mathcal{C}_q$ and an $[n, t; M'_1, \ldots, M'_t]_{q'}$ t-write WOM-code $\mathcal{C}_{q'}$, then there exists an $[n, t; M_1 \cdot M'_1, \ldots, M_t \cdot M'_t]_{q \cdot q'}$ t-write WOM-code.*

Proof. Assume that t encoding and decoding maps of the WOM-code $\mathcal{C}_q$ are denoted by $\mathcal{E}_{q,j}, \mathcal{D}_{q,j}$, for all j where $1 \leq j \leq t$. Similarly, assume that t encoding and decoding maps of the WOM-code $\mathcal{C}_{q'}$ are denoted by $\mathcal{E}_{q',j}, \mathcal{D}_{q',j}$, for all j where $1 \leq j \leq t$. We construct an $[n, t; M_1 \cdot M'_1, \ldots, M_t \cdot M'_t]_{q \cdot q'}$ t-write WOM-code which we denote by $\mathcal{C}_{q \cdot q'}$ with encoding and decoding maps $\mathcal{E}_{q \cdot q',j}, \mathcal{D}_{q \cdot q',j}$, as follows.

On the j^{th} write, $1 \leq j \leq t$, the input to the map $\mathcal{E}_{q \cdot q',j}$ is $(m_j, \boldsymbol{c}_{j-1})$ where $m_j \in \mathbb{Z}_{M_j \cdot M'_j}$ and $\boldsymbol{c}_{j-1} = (c_1, \ldots, c_n) \in \mathbb{Z}^n_{q \cdot q'}$ is a cell-state vector. Let $v_{j,1} \equiv m_j \mod M_j$ and $v_{j,2} = \lfloor \frac{m_j}{M_j} \rfloor$.

Define $\boldsymbol{z}^{(1)} \equiv \boldsymbol{c}_{j-1} \mod q$ and $\boldsymbol{z}^{(2)} = \lfloor \frac{\boldsymbol{c}_{j-1}}{q} \rfloor$. Let

$$\boldsymbol{u}^{(1)} = \mathcal{E}_{q,j}(v_{j,1}, \boldsymbol{z}^{(1)}), \text{ and}$$
$$\boldsymbol{u}^{(2)} = \mathcal{E}_{q',j}(v_{j,2}, \boldsymbol{z}^{(2)}),$$

where $\boldsymbol{u}^{(1)} \in \mathbb{Z}_q^n$ and $\boldsymbol{u}^{(2)} \in \mathbb{Z}_{q'}^n$. Then, the output of the encoding map is defined as

$$\mathcal{E}_{q\cdot q',j}(m_j, \boldsymbol{c}_{j-1}) = \boldsymbol{u}^{(1)} + q \cdot \boldsymbol{u}^{(2)}.$$

Since $\mathcal{C}_q, \mathcal{C}_{q'}$ are WOM-codes, then $\boldsymbol{u}^{(1)} \geq \boldsymbol{z}^{(1)}, \boldsymbol{u}^{(2)} \geq \boldsymbol{z}^{(2)}$ and hence $\mathcal{E}_{q\cdot q',j}(m_j, \boldsymbol{c}_{j-1}) \geq \boldsymbol{c}_{j-1}$.

Similarly, for $1 \leq j \leq t$, we define the decoding map as follows. Let $\boldsymbol{c}_j \in \mathbb{Z}_{q\cdot q'}^n$ be the cell-state vector. Let $\hat{\boldsymbol{u}}^{(1)} = \boldsymbol{c}_j \mod q$ and $\hat{\boldsymbol{u}}^{(2)} = \lfloor \frac{\boldsymbol{c}_j}{q} \rfloor$. Then

$$\hat{v}_{j,1} = \mathcal{D}_{q,j}(\hat{\boldsymbol{u}}^{(1)}),$$
$$\hat{v}_{j,2} = \mathcal{D}_{q',j}(\hat{\boldsymbol{u}}^{(2)}).$$

The output of the decoding map is defined as

$$\mathcal{D}_{q\cdot q',j}(\boldsymbol{c}_j) = \hat{v}_{j,1} + M_j \cdot \hat{v}_{j,2}.$$

It is straightforward to show that $\hat{v}_{j,1} = v_{j,1}$ and $\hat{v}_{j,2} = v_{j,2}$, so that $\mathcal{D}_{q\cdot q',j}(\boldsymbol{c}_j) = m_j$ as desired. □

2.2.2 Ladder Construction

The key idea behind the ***Ladder Construction*** introduced in the next lemma (Lemma 4 below) is the following. On the first write we only use the symbols $\{0, \ldots, L \cdot q - 1\}$ where L, q are positive integers and $L \geq 2, q \geq 2$. On the second write we only use the symbols

$\{L-1,\ldots,(L\cdot q-1)+(L-1)\}$, and so on until the last (or t^{th} write) where we use symbols $\{(t-1)\cdot(L-1),\ldots,(L\cdot q-1)+(t-1)\cdot(L-1)\}$. We divide up the $L\cdot q$ available symbols at each generation into equally sized blocks of length L. At each generation we use a q-ary WOM-code to select, for each cell, these L consecutive levels to write into. Each of the symbols can then take on any of the L values in that block and so this allows us to encode an additional message vector with symbols from $\mathbb{Z}_L$.

This construction provides high rate codes for the case where the symbol size is large whereas the Expansion Construction produces high rate codes for the case where the symbol size is small. We note that the construction presented here achieves a higher rate than the version introduced in our previous work [GYDSVW11]. In the lemma below, we refer to the all-ones vector as $\mathbf{1}_n$. The Ladder Construction is described in a slightly informal manner in the next lemma (Lemma 4) for the two-write case. Suppose $\boldsymbol{v}=(v_1,\ldots,v_n)\in\mathbb{Z}_q^n$. Then $\lfloor\boldsymbol{v}\rfloor=(\lfloor v_1\rfloor,\ldots,\lfloor v_n\rfloor)$.

Lemma 4. *Let $\mathcal{C}_q$ be an $[n,2;M_1,M_2]_q$ 2-write WOM-code and let L be an integer where $L\geq 2$. Then, there exists an $[n,2;M_1\cdot L^n,M_2\cdot L^n]_{L\cdot(q+1)-1}$ 2-write WOM-code.*

Proof. We prove the result by describing the encoding and decoding procedure. For shorthand, let $q'=L\cdot(q+1)-1$ and let $\mathcal{C}_{q'}$ be the $[n,2;M_1\cdot L^n,M_2\cdot L^n]_{q'}$ 2-write WOM-code referred to in the lemma. For the first write, we pick a first generation codeword, say $\boldsymbol{c}'\in\mathbb{Z}_q^n$ from $\mathcal{C}_q$ as well as a word $\boldsymbol{w}_1\in\mathbb{Z}_L^n$ to be written. We define $\boldsymbol{c}_1=\boldsymbol{c}'\cdot L+\boldsymbol{w}_1$ to be a first generation codeword in $\mathcal{C}_{q'}$. Clearly, a decoder for $\mathcal{C}_{q'}$ can recover the information $\boldsymbol{c}'$, $\boldsymbol{w}_1$ from $\boldsymbol{c}_1$ (by simply taking modulo operations or floor operations). Furthermore, since $\boldsymbol{c}'\in\mathcal{C}_q$ and the number of first generation messages for the code $\mathcal{C}_q$ is M_1, the number of first generation messages for the code $\mathcal{C}_{q'}$ on the first generation is $M_1\cdot L^n$.

For the second write, we first recover the vector $\boldsymbol{c}'=\lfloor\frac{\boldsymbol{c}_1}{L}\rfloor$. We now choose a second generation codeword $\boldsymbol{c}''$ from $\mathcal{C}_q$ provided the cell-state vector $\boldsymbol{c}'$ so that $\boldsymbol{c}''\geq\boldsymbol{c}'$ (since $\mathcal{C}_q$ is a WOM-code). Similar to before, we choose another word $\boldsymbol{w}_2\in\mathbb{Z}_L^n$ to also be stored into memory. The state of the memory is now updated to $\boldsymbol{c}_2=(\boldsymbol{c}''+\mathbf{1}_n)\cdot L+\boldsymbol{w}_2-\mathbf{1}_n$. Since

$\boldsymbol{c}'' \geq \boldsymbol{c}'$, we have that $\boldsymbol{c}_2 = (\boldsymbol{c}'' + \mathbf{1}_n) \cdot L + \boldsymbol{w}_2 - \mathbf{1}_n \geq \boldsymbol{c}' \cdot L + \boldsymbol{w}_1 = \boldsymbol{c}_1$ so that the WOM-constraint is satisfied. In this case the decoder can determine $\boldsymbol{c}''$ since $\boldsymbol{c}'' = \lfloor \frac{\boldsymbol{c}_2 + \mathbf{1}_n}{L} \rfloor - \mathbf{1}_n$ and $\boldsymbol{w}_2 = \boldsymbol{c}_2 + \mathbf{1}_n (\mathrm{mod} L)$. Since $\boldsymbol{w}_2 \in \mathbb{Z}_L^n$ and the number of second generation messages for the code $\mathcal{C}_q$ is M_2, the number of second generation messages for the code $\mathcal{C}_{q'}$ is $M_2 \cdot L^n$. $\square$

A more general (and more formal) presentation of the Ladder code construction is below.

Lemma 5. *Let $\mathcal{C}_q$ be an $[n, t; M_1, \ldots, M_t]_q$ t-write WOM-code and let L be an integer where $L \geq 2$. Then, there exists an $[n, t; M_1 \cdot L^n, \ldots, M_t \cdot L^n]_{L(q+t-1)-(t-1)}$ t-write WOM-code.*

Proof. We let $\mathcal{E}_{q,j}, \mathcal{D}_{q,j}$ for all j where $1 \leq j \leq t$ be the encoding and decoding maps for the $[n, t; M_1, \ldots, M_t]_q$ t-write WOM-code $\mathcal{C}_q$. Let $q' = L(q + t - 1) - (t - 1)$. We construct an $[n, t; M_1 \cdot L^n, \ldots, M_t \cdot L^n]_{q'}$ t-write WOM-code denoted $\mathcal{C}_{q'}$, with encoding and decoding maps $\mathcal{E}_{q',j}, \mathcal{D}_{q',j}$ as follows.

The input to the encoding map $\mathcal{E}_{q',j}$ on the j^{th} write, $1 \leq j \leq t$, is $(m_j, \boldsymbol{c}_{j-1})$ where $m_j \in \mathbb{Z}_{M_j \cdot L^n}$ and $\boldsymbol{c}_{j-1} \in \mathbb{Z}_{q'}^n$ is a cell-state vector. Suppose

$$w_{j,0} \equiv m_j \mod M_j, \tag{2.7}$$

and

$$(w_{j,1}, \ldots, w_{j,n}) = \phi_{L,n}^{-1} \left(\left\lfloor \frac{m_j}{M_j} \right\rfloor \right) \in \mathbb{Z}_L^n. \tag{2.8}$$

Recall that for the case where $j = 1$, $\mathcal{E}_{q,1}$ ignores the cell-state vector. For $2 \leq j \leq t$, let

$$\boldsymbol{z} = \left\lfloor \frac{\boldsymbol{c}_{j-1} + (j-2) \cdot \mathbf{1}_n}{L} \right\rfloor - (j-2) \cdot \mathbf{1}_n. \tag{2.9}$$

We have

$$\boldsymbol{u} = \mathcal{E}_{q,j}(w_{j,0}, \boldsymbol{z}) + (j-1) \cdot \mathbf{1}_n, \tag{2.10}$$

where $\boldsymbol{u} \in \mathbb{Z}_{q'}^n$.

The cell-state vector is updated to

$$\boldsymbol{c}_j = L \cdot \boldsymbol{u} + (w_{j,1}, \ldots, w_{j,n}) - (j-1) \cdot \mathbf{1}_n. \tag{2.11}$$

Notice that $\boldsymbol{z}$ is equal to the output of the encoder $\mathcal{E}_{q,j-1}$ for $j \geq 2$. This can be verified by substituting (2.10), (2.11) into (2.9). The terms involving a multiple of $\mathbf{1}_n$ in (2.9), (2.10), and (2.11) ensure we are in the correct group of Lq levels to write into. Since $\mathcal{E}_{q,j}$ is the encoder for the WOM-code $\mathcal{C}_q$, $\mathcal{E}_{q,j}(w_{j,0}, \boldsymbol{z}) \geq \boldsymbol{z}$ in (2.10) and the value of j in (2.10) and (2.11) strictly increases upon every write. Thus, from (2.9), (2.10) and (2.11), $\boldsymbol{c}_j \geq \boldsymbol{c}_{j-1}$, and so the WOM-constraint is satisfied.

The decoding map $\mathcal{D}_{q',j} : \mathbb{Z}_{q'}^n \to \mathbb{Z}_{M_j \cdot L^n}$ for $1 \leq j \leq t$ is defined as follows. Let $\boldsymbol{c}_j = (c_1, \ldots, c_n) \in \mathbb{Z}_{q'}^n$ be the cell-state vector after the j^{th} write and let

$$\boldsymbol{y} = \left\lfloor \frac{\boldsymbol{c}_j + (j-1) \cdot \mathbf{1}_n}{L} \right\rfloor - (j-1) \cdot \mathbf{1}_n. \tag{2.12}$$

Substituting $\boldsymbol{c}_j$ from (2.11) into (2.12) shows that $\boldsymbol{y} = \boldsymbol{u} - (j-1) \cdot \mathbf{1}_n = \mathcal{E}_{q,j}(w_{j,0}, \boldsymbol{z})$. Thus,

$$\mathcal{D}_{q,j}(\boldsymbol{y}) = \mathcal{D}_{q,j}(\mathcal{E}_{q,j}(w_{j,0}, \boldsymbol{z})) = w_{j,0}. \tag{2.13}$$

Let $\hat{w}_{j,i} \equiv (c_i + (j-1)) \mod L$. Finally, we define $\mathcal{D}_{q',j}$ as

$$\mathcal{D}_{q',j}(\boldsymbol{c}_j) = \mathcal{D}_{q,j}(\boldsymbol{y}) + \phi_{L,n}(\hat{w}_{j,1}, \ldots, \hat{w}_{j,n}) \cdot M_j.$$

According to (2.11), we have that for $1 \leq i \leq n$, $w_{j,i} \equiv (c_i + (j-1)) \mod L$ and thus $w_{j,i} = \hat{w}_{j,i}$ so that, using (2.13),

$$\mathcal{D}_{q',j}(\boldsymbol{c}_j) = w_{j,0} + \phi_{L,n}(\hat{w}_{j,1}, \ldots, \hat{w}_{j,n}) \cdot M_j = m_j.$$

□

For shorthand, we refer to the construction presented in Lemma 5 as the Ladder Construction. The code $\mathcal{C}_q$ in the statement of the lemma will be referred to as the base WOM-code. As in the previous subsection, we illustrate the encoding process for a code created according to the Ladder Construction with an example.

Example 2. *Let $\mathcal{C}_2$ be the $[3, 2; 4, 4]_2$ Rivest-Shamir two-write WOM-code from Table 1. We use $\mathcal{C}_2$ as the base code to construct a $[3, 2; 4 \cdot 3^3, 4 \cdot 3^3]_8$ two-write WOM-code $\mathcal{C}_8$ over an alphabet with symbols from $\mathbb{Z}_8$ using the Ladder Construction from Lemma 5. In this case, the blocks chosen by the code $\mathcal{C}_2$ will be of size $L = 3$.*

Suppose that on the first write ($j = 1$) we choose the codeword $\boldsymbol{c}' = (0, 0, 1)$ from $\mathcal{C}_2$ as well as the word $\boldsymbol{w}_1 = (0, 0, 1)$. We store the first generation codeword $\boldsymbol{c}_1 = (0, 0, 1) \cdot 3 + (0, 0, 1) = (0, 0, 4)$. For the second write, suppose $\boldsymbol{c}'' = (0, 1, 1)$ and $\boldsymbol{w}_2 = (0, 0, 0)$. Under this setup, we store the second generation codeword $\boldsymbol{c}_2 = ((0, 1, 1) + (1, 1, 1)) \cdot 3 + (0, 0, 0) - (1, 1, 1) = (2, 5, 5)$. Observe that the WOM-constraint is satisfied.

The following corollary holds since at generation j (where $1 \leq j \leq t$), we can store $M_j \cdot L^n$ messages, where M_j is the number of messages code $\mathcal{C}_q$ in Lemma 5 can encode at generation j.

Corollary 3. *If $\mathcal{R}_B$ is the sum-rate of the base WOM-code $\mathcal{C}_q$ in Lemma 5, then the sum-rate of the WOM-code $\mathcal{C}_{L(q+t-1)-(t-1)}$ created according to Lemma 5 is $t \cdot \log_2 L + \mathcal{R}_B$.*

We note that in the short code length regime, a code created according to Lemma 5 in many cases achieves the highest known rate for a fixed code length. For example, for the case where $n = 3$ and $L = 3$, the Rivest and Shamir code produces a code over $\mathbb{Z}_8$ of length 3 with sum-rate 4.50.

By way of comparison, we note that 3-cell WOM-code constructions were studied recently in [CY12] that leveraged the tilings discussed in [SE13]. However, one potential drawback of these constructions is that the choice of parameters is limited. More concretely, the length-3

codes proposed in [CY12] guarantee for an integer $c > 1$, $1+c+c^2$ writes, which implies that the resulting codebooks guarantee at least 7 writes. In contrast, the Ladder Construction presented in this subsection provides a straightforward way to convert existing high-rate t-write binary codes into t-write non-binary codes for small t.

Both the Expansion Construction and the Ladder Construction are capable of producing short WOM-codes over non-binary alphabets. Furthermore, the constructions result in t-write WOM-codes over many possible alphabets (where $t \geq 2$). In the next section, we describe a two-write WOM-code construction that, in many instances, improves upon existing code constructions.

2.3 Spider Codes

In this section, we propose a family of two-write 4-ary WOM-codes known as ***Spider Codes***. We chose this name due to the web-like mappings between binary and non-binary codebooks. Similar to the Expansion construction, Spider Codes are 4-ary codes that are the result of mapping binary codebooks into higher alphabets. The difference between Spider Codes and the Expansion Construction is that in the Expansion Construction the mappings are to independent binary WOM-codes. The constituent codes used by Spider Codes are very much dependent on each other in a manner analogous to the three-write WOM-code in [YS12]. However, unlike the three-write binary WOM-code in [YS12], a set of auxiliary two-write codebooks are used. For carefully chosen parameters, Spider Codes can achieve a sum-rate within 0.03 of capacity. We provide a more detailed comparison of our construction with the state of the art following the presentation of the code construction.

We first introduce some notation.

Definition 3. *We say that* $\boldsymbol{p}_q = (p_0, \ldots, p_{q-1})$ *is a* ***symbol-distribution vector*** *if for* $0 \leq i \leq q-1$, $0 < p_i < 1$ *and* $\sum_{i=0}^{q-1} p_i = 1$.

For the remainder of this section we assume $q = 4$. Since $q = 4$, we drop the subscript

on $\boldsymbol{p}$ and refer to $\boldsymbol{p}_4$ as $\boldsymbol{p}$. For a symbol-distribution vector $\boldsymbol{p}$ and a positive integer n, let

$$\mathcal{C}(\boldsymbol{p}, n) = \{\boldsymbol{x} =(x_1, \ldots, x_n) \in \mathbb{Z}_4^n : \\ 0 \leq a \leq 3, |\{i : x_i = a\}| = p_a n\}, \tag{2.14}$$

where we assume that $p_i n$ is integer-valued for $0 \leq i \leq 3$.

We now proceed with an informal description of the code construction. Assume the code length is $n + K$ for positive integers n, K. The first n symbols can be thought of as storing the first and second generation codewords. Additionally, K symbols are appended to the length n vectors to store metadata, which will specify the choice of encoding maps to be defined in Section 2.3.2. We call the appended symbols the ***dedicated codeword suffix***. We describe the code construction in terms of vectors of length n. It is assumed there is some additional amount of space, in the dedicated codeword suffix, that can be used to store metadata.

We make use of the following theorem from [SHP12a] where the function $\mathcal{H}$ denotes the entropy function. We assume here and the rest of the chapter that the terms $2^{n(\mathcal{H}(p)-\epsilon)}, 2^{n(1-p-\epsilon)}$ are integer-valued.

Theorem 1. *(cf. [SHP12a]) For any $0 < p < 1$ and $\epsilon > 0$, there exists an $[n, 2; 2^{n(\mathcal{H}(p)-\epsilon)}, 2^{n(1-p-\epsilon)}]_2$ WOM-code.*

The proof of Theorem 1 shows that, given a two-write code that uses at most pn symbols in the first generation, it is possible in the second generation to encode at a rate $1 - p - \epsilon$ (by programming only the symbols with value 0 in the first generation) using the second generation encoder of a capacity-achieving two-write binary WOM-code. Using the tools from [SHP12a], the following lemma follows. A similar lemma can also be found in [YS12].

Lemma 6. *(cf. [YS12]) For any $\epsilon > 0$, there exists a 2-write (binary) WOM code $\mathcal{C}$ of length n such that on the first write at most pn symbols are programmed, and on the second write $\mathcal{C}$ achieves rate $(1 - p - \epsilon)$.*

We begin in Section 2.3.1 with a description of an auxiliary 4-ary WOM-code $\tilde{\mathcal{C}}_4$ that has first generation codewords from $\mathcal{C}(\boldsymbol{p}, n)$. Using the description of the code $\tilde{\mathcal{C}}_4$, the Spider Code Construction is presented in Section 2.3.2. An example of a finite length Spider Code is presented in Section 2.3.3.

2.3.1 The Auxiliary Code $\tilde{\mathcal{C}}_4$

The purpose of this subsection is to describe a two-write WOM-code over $\mathbb{Z}_4$, denoted as $\tilde{\mathcal{C}}_4$. The principal tool used to encode second generation codewords for $\tilde{\mathcal{C}}_4$ will be a second generation encoder from a binary two-write code from Lemma 6. We first describe some useful maps. These maps are displayed later in this section in the form of a table (see Table 2.3).

Let $\psi_a : \mathbb{Z}_4 \times \mathbb{Z}_2 \to \mathbb{Z}_4$ be defined so that for $c \in \mathbb{Z}_4$ and $d \in \mathbb{Z}_2$, we have

$$\psi_a(c, d) = ((c + d(\mathrm{mod}2)) + c)\,(\mathrm{mod}4).$$

Let $\psi_b : \mathbb{Z}_4 \times \mathbb{Z}_2 \to \mathbb{Z}_4$ be defined so that for $c \in \mathbb{Z}_4$ and $d \in \mathbb{Z}_2$ $\psi_2(c, d) = \psi_1(c, d)$ when $c \geq 2$, and for $c < 2$

$$\psi_b(0, 0) = 1, \psi_b(0, 1) = 0, \psi_b(1, 0) = 1, \psi_b(1, 1) = 3.$$

We define $\hat{\psi}_a^{-1} : \mathbb{Z}_4 \to \mathbb{Z}_2, \hat{\psi}_b^{-1} : \mathbb{Z}_4 \to \mathbb{Z}_2$ as follows

$$\hat{\psi}_1^{-1}(a) = a \bmod 2,$$
$$\hat{\psi}_2^{-1}(0) = 1, \hat{\psi}_2^{-1}(1) = 0, \hat{\psi}_2^{-1}(2) = 0, \hat{\psi}_2^{-1}(3) = 1.$$

We note that the map $\hat{\psi}_\ell^{-1}$ for $\ell \in \{a, b\}$ behaves similarly to an inverse. In particular, for any $(c, d) \in \mathbb{Z}_4 \times \mathbb{Z}_2$ where $(c, d) \neq (3, 0)$ we have $\hat{\psi}_\ell^{-1}(\psi_\ell(c, d)) = d$.

If the input to the maps $\psi_a, \psi_b, \hat{\psi}_a^{-1}, \hat{\psi}_b^{-1}$ are vectors, then the map is simply applied

component-wise. We now have the main result. As will be described shortly, the key property of the code $\tilde{\mathcal{C}}_4$ described in the next lemma (Lemma 7) is that any second generation codeword in $\tilde{\mathcal{C}}_4$ has a small number of 2 and 3 symbols. Let $\mathcal{C}_2 = [n, 2; M_1, M_2]_2$ be a two-write binary WOM-code (as provided by Lemma 6) such that $M_1 = 2^{n(\mathcal{H}(p_3)-\epsilon)}$ and $M_2 = 2^{n(1-p_3-\epsilon)}$, where M_1 and M_2 are assumed to be integer-valued.

Lemma 7. *Let $\boldsymbol{p} = (p_0, p_1, p_2, p_3)$ be a symbol-distribution vector. For any $\epsilon > 0$ and n sufficiently large, there exists an*
$[n+1, 2; 2^{n(\mathcal{H}(\boldsymbol{p})-\epsilon)}, 2^{n(1-p_3-\epsilon)}]_4$ WOM-code $\tilde{\mathcal{C}}_4$.

Proof. We prove the lemma by describing the encoding and decoding procedures. A first generation codeword, say $\tilde{\boldsymbol{c}}_1 = (\tilde{c}_{1,1}, \ldots, \tilde{c}_{1,n})$ is chosen from $\mathcal{C}(\boldsymbol{p}, n)$ for some symbol-distribution vector $\boldsymbol{p}$. The encoding and decoding maps are straightforward. We assume there is one additional symbol that will constitute the dedicated codeword suffix so that the length of the code $\tilde{\mathcal{C}}_4$ is $n+1$.

For the second generation, we then choose a second generation codeword from $\mathcal{C}_2$ provided the cell-state vector $\boldsymbol{z} = (z_1, \ldots, z_n) \in \mathbb{Z}_2^n$ where for all i where $1 \leq i \leq n$, $z_i = 1$ if $\tilde{c}_{1,i} = 3$ and $z_i = 0$ otherwise. The symbol 3 is important since once a symbol reaches the value 3, the symbol can no longer be used to store more information. Suppose the second generation codeword chosen from the binary codebook $\mathcal{C}_2$ is $\boldsymbol{u} = (u_1, \ldots, u_n)$ (where $\boldsymbol{u} \geq \boldsymbol{z}$).

We now update the state of the memory from $\tilde{\boldsymbol{c}}_1$ to $\tilde{\boldsymbol{c}}_2$ using the map ψ_ℓ where $\ell \in \{a, b\}$. If $|\{i : \tilde{c}_{1,i} = 1, u_i = 1\}| \geq \{i : \tilde{c}_{1,i} = 1, u_i = 0\}|$, then $\psi_\ell = \psi_a$ and $\psi_\ell = \psi_b$ otherwise. Finally the state of the memory is updated from $\tilde{\boldsymbol{c}}_1 = (\tilde{c}_{1,1}, \ldots, \tilde{c}_{1,n})$ to $\tilde{\boldsymbol{c}}_2 = (\tilde{c}_{2,1}, \ldots, \tilde{c}_{2,n})$ so that for all i where $1 \leq i \leq n$, we have $\tilde{c}_{2,i} = \psi_\ell(\tilde{c}_{1,i}, u_i)$. The value of ℓ is then encoded into the dedicated codeword suffix. We note that $\tilde{\boldsymbol{c}}_2 \geq \tilde{\boldsymbol{c}}_1$ since it can be verified that for $\ell \in \{a, b\}$ and any second generation codeword $\boldsymbol{u} \in \mathcal{C}_2$ (where $\boldsymbol{u} \geq \boldsymbol{z}$), we have $\psi_\ell(\tilde{\boldsymbol{c}}_1, \boldsymbol{u}) \geq \tilde{\boldsymbol{c}}_1$. Furthermore, the codeword $\boldsymbol{u}$ can be recovered from $\tilde{\boldsymbol{c}}_1$ by first determining ℓ from the dedicated codeword suffix and then noting that $\hat{\psi}_\ell^{-1}(\tilde{\boldsymbol{c}}_2) = \boldsymbol{u}$.

We have illustrated the encoding and decoding maps for a two-write WOM-code that

can represent $|\mathcal{C}(\boldsymbol{p}, n)| = 2^{n(\mathcal{H}(\boldsymbol{p})-\epsilon)}$ messages in the first generation. Since in the second generation, the encoder for $\tilde{\mathcal{C}}_4$ can choose any message of the $M_2 = 2^{n(1-p_3-\epsilon)}$ messages from $\mathcal{C}_2$ (by selecting a second generation codeword from $\mathcal{C}_2$), the statement in the lemma follows. □

Let $\tilde{\mathcal{E}}_{4,2}/\tilde{\mathcal{D}}_{4,2}$ be the second generation encoder/decoder for the code $\tilde{\mathcal{C}}_4$ as described in Lemma 7. The following claims will be useful in the next subsection.

Claim 1. *Let $\tilde{\mathcal{C}}_4$ be the $[n+1, 2; 2^{n(\mathcal{H}(\boldsymbol{p})-\epsilon)}, 2^{n(1-p_3-\epsilon)}]_4$ code from Lemma 7. Let $\ell \in \{a, b\}$. If $\boldsymbol{c}, \boldsymbol{c}'$ are two second generation codewords in $\tilde{\mathcal{C}}_4$ encoded using ψ_ℓ where $\hat{\psi}_\ell^{-1}(\boldsymbol{c}) = \hat{\psi}_\ell^{-1}(\boldsymbol{c}')$, then $\tilde{\mathcal{D}}_{4,2}(\boldsymbol{c}) = \tilde{\mathcal{D}}_{4,2}(\boldsymbol{c}')$.*

Claim 2. *Let $\tilde{\mathcal{C}}_4$ be the two-write WOM-code of length $n+1$ from Lemma 7. Then, for any second generation codeword $\tilde{\boldsymbol{c}}_2 = (\tilde{c}_{2,1}, \ldots, \tilde{c}_{2,n}) \in \tilde{\mathcal{C}}_4$, there exists a parameter $p_s \in \mathbb{R}$ where $0 \leq p_s \leq 1$ such that the following holds:*

1. $|\{i : \tilde{c}_{2,i} = 0\}| = p_s p_0 n$,
2. $|\{i : \tilde{c}_{2,i} = 1\}| \geq \frac{p_1}{2} n + (1 - p_s) p_0 n$.

Proof. Recall, from the proof of Lemma 7, that the output of $\tilde{\mathcal{E}}_{4,2}$ is $\psi_\ell(\tilde{\boldsymbol{c}}_1, \boldsymbol{u})$ where $\ell \in \{a, b\}$, $\tilde{\boldsymbol{c}}_1$ is a first generation codeword in $\tilde{\mathcal{C}}_4$, and $\boldsymbol{u}$ is a second generation codeword in the binary codebook $\mathcal{C}_2$. Let $p_s n = |\{i : \tilde{c}_{1,i} = \tilde{c}_{2,i} = 0\}|$. Because $\psi_\ell(0, d) = \{0, 1\}$ for any $d \in \mathbb{Z}_2$, we have that if $p_s n = |\{i : \tilde{c}_{1,i} = \tilde{c}_{2,i} = 0\}|$, then $|\{i : \tilde{c}_{1,i} = 0, \tilde{c}_{2,i} = 1\}| = (1 - p_s)n$. Since $p_0 n = |\{i : \tilde{c}_{1,n} = 0\}|$, then $|\{i : \tilde{c}_{2,i} = 0\}| = p_s p_0 n$ and $|\{i : \tilde{c}_{2,i} = 1, \tilde{c}_{1,i} = 0\}| = (1 - p_s) p_0 n$.

Recall from the proof of Lemma 7, the parameter ℓ is chosen so that if $|\{i : \tilde{c}_{1,i} = 1, u_i = 1\}| \geq |\{i : \tilde{c}_{1,i} = 1, u_i = 0\}|$, then $\psi_\ell = \psi_a$ and $\psi_\ell = \psi_b$ otherwise. Because $\psi_a(1, 1) = \psi_b(1, 0) = 1$, the choice of ℓ ensures that $|\{i : \tilde{c}_{1,i} = 1, \tilde{c}_{2,i} = 1\}| \geq |\{i : \tilde{c}_{1,i} = 1, \tilde{c}_{2,i} > 1\}|$. Since $p_1 n = |\{i : \tilde{c}_{1,i} = 1\}|$, we can conclude that $|\{i : \tilde{c}_{1,i} = 1, \tilde{c}_{2,i} = 1\}| \geq \frac{p_1 n}{2}$. Combining this with $|\{i : \tilde{c}_{2,i} = 1, \tilde{c}_{1,i} = 0\}| = (1 - p_s) p_0 n$ from the previous paragraph, gives the result in the statement of the lemma. □

In addition to the code $\tilde{\mathcal{C}}_4$, we will require one more tool to describe the Spider Code construction known as the binary pair collection. More specifically, the ***binary pair collection*** is a collection of binary WOM-codes of the form $[n, 2; 2^{n(\mathcal{H}(p)-\epsilon)}, 2^{n(1-p-\epsilon)}]_2$ for varying values of p where $0 < p < 1$ and $\epsilon > 0$. In particular, we will assume the binary pair collection consists of the codes described in Lemma 6. For p_0, p_1 ($0 < p_0, p_1 < 1$ and $p_0 + p_1 < 1$) and a positive integer r, we interpret the binary pair collection as a vector $\mathbf{B}(p_0, p_1, r) = (\boldsymbol{b}_0, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_r)$, such that for $0 \leq i \leq r$, the i^{th} entry of $\mathbf{B}(p_0, p_1, r)$ is

$$\boldsymbol{b}_i = ([n, 2; M_1^{(i,1)}; M_2^{(i,1)}]_2, [n, 2; M_1^{(i,2)}, M_2^{(i,2)}]_2), \tag{2.15}$$

where $M_1^{(i,1)} = 2^{n(\mathcal{H}(1-p_0\frac{i}{r})-\epsilon)}$, $M_2^{(i,1)} = 2^{n(p_0\frac{i}{r}-\epsilon)}$, $M_1^{(i,2)} = 2^{n(\mathcal{H}(\frac{r+1}{r}-\frac{p_1}{2}-(1-\frac{i}{r})p_0)-\epsilon)}$, and $M_2^{(i,2)} = 2^{n(\frac{p_1}{2}+(1-\frac{i}{r})p_0-\frac{1}{r}-\epsilon)}$ and we assume that $M_2^{(0,1)} = 1$ so that $M_2^{(0,1)}$ is an integer.[1]

The function g will be used to determine which binary codebook pair to use to encode more information. Let $g : [0, 1] \times [0, 1] \times \mathbb{Z}^+ \to \mathbb{N}$ be such that

$$g(p', p_0, r) = \max\{i : p' \geq p_0\frac{i}{r}\}. \tag{2.16}$$

The main idea behind the Spider Codes is contained within the following claim. The claim states that if we know how many symbols with value 0 are in a second generation codeword $\tilde{\boldsymbol{c}}_2 \in \tilde{\mathcal{C}}_4$ (where $\tilde{\mathcal{C}}_4$ is a code from Lemma 7), then we can determine a lower bound on the number of symbols in $\tilde{\boldsymbol{c}}_2$ with a value 1.

In the following claim, $\tilde{\mathcal{C}}_4$ is an $[n+1, 2; 2^{n(\mathcal{H}(\boldsymbol{p})-\epsilon)}, 2^{n(1-p_3-\epsilon)}]_4$ WOM-code from Lemma 7, where we assume the codewords are vectors of length n and there is an additional symbol appended to every codeword (known as the dedicated codeword suffix of size $K = 1$).

Claim 3. *Let r be a positive integer. For any second generation codeword $\tilde{\boldsymbol{c}}_2 = (\tilde{c}_{2,1}, \ldots, \tilde{c}_{2,n}) \in$*

[1] We assume that p_0, p_1 are chosen so that for all i, $0 < \frac{p_1}{2} + (1 - \frac{i}{r})p_0 - \frac{1}{r} - \epsilon < 1$. Here, as elsewhere, we assume $M_1^{(i,1)}, M_2^{(i,1)}, M_1^{(i,2)}, M_2^{(i,2)}$ are integer-valued.

$\tilde{\mathcal{C}}_4$, *if* $g(\frac{|\{i:\tilde{c}_{2,i}=0\}|}{n}, p_0, r) = k$, *then* $\frac{|\{i:\tilde{c}_{2,i}=1\}|}{n} \geq \frac{p_1}{2} + p_0(1 - \frac{k}{r}) - \frac{1}{r}$.

Proof. Let $p' = \frac{|\{i:\tilde{c}_{2,i}=0\}|}{n}$ where $\tilde{\boldsymbol{c}}_2$ is a second generation codeword from $\tilde{\mathcal{C}}_4$. From Claim 2, there exists a parameter $p_s \in \mathbb{R}$ where $0 \leq p_s \leq 1$ such that $\frac{|\{i:\tilde{c}_{2,i}=0\}|}{n} = p_0 p_s$ and $\frac{|\{i:\tilde{c}_{2,i}=1\}|}{n} \geq \frac{p_1}{2} + (1 - p_s)p_0$. If $k = g(p', p_0, r)$ then $p' = p_0 p_s < p_0 \frac{k}{r} + \frac{1}{r}$, which implies $p_s < \frac{k}{r} + \frac{1}{p_0 r}$ so that

$$\frac{|\{i : \tilde{c}_{2,i} = 1\}|}{n} \geq \frac{p_1}{2} + p_0(1 - \frac{k}{r}) - \frac{1}{r}.$$

□

The following corollary follows from Claim 3. Recall that for a two-write WOM-Code $\mathcal{C}_q$, the rate of the second generation is denoted $\mathcal{R}_2(\mathcal{C}_q)$. Recall that $\tilde{\mathcal{C}}_4$ has first generation codewords from $\mathcal{C}(\boldsymbol{p}, n)$ where $\boldsymbol{p} = (p_0, p_1, p_2, p_3)$.

Corollary 4. *Let* $\tilde{\boldsymbol{c}}_2$ *be a second generation codeword from the code* $\tilde{\mathcal{C}}_4$ *from Lemma 7. Suppose we have the binary pair collection* $\mathbf{B}(p_0, p_1, r) = (\mathbf{b}_0, \ldots, \mathbf{b}_r)$ *where* r *is a positive integer. Let* $k = g(p', p_0, r)$ *where* $p' = \frac{|\{i:\tilde{c}_{2,i}=0\}|}{n}$. *Then, the* k*-th entry of* $\mathbf{B}(p_0, p_1, r)$*, denoted as* $(\mathcal{C}_2^{(0)}, \mathcal{C}_2^{(1)})$*, is such that:*

1. $\frac{|\{i:\tilde{c}_{2,i}=0\}|}{n} \geq \mathcal{R}_2(\mathcal{C}_2^{(0)}) + \epsilon$, *and*
2. $\frac{|\{i:\tilde{c}_{2,i}=1\}|}{n} \geq \mathcal{R}_2(\mathcal{C}_2^{(1)}) + \epsilon$.

Using the binary pair collection along with the auxiliary code $\tilde{\mathcal{C}}_4$, we describe the Spider Code construction in the next subsection.

2.3.2 Spider Code Construction

We begin this section by introducing some mappings that will be used for the code construction. Afterwards, the encoding and decoding procedures for a Spider Code are presented. For shorthand, we refer to the resulting Spider Code as $\mathcal{C}_4$.

Let $\psi_{a,0} : \mathbb{Z}_4 \times \mathbb{Z}_2 \to \mathbb{Z}_4$, $\psi_{b,0} : \mathbb{Z}_4 \times \mathbb{Z}_2 \to \mathbb{Z}_4$, $\psi_{a,1} : \mathbb{Z}_4 \times \mathbb{Z}_2 \to \mathbb{Z}_4$, $\psi_{b,1} : \mathbb{Z}_4 \times \mathbb{Z}_2 \to \mathbb{Z}_4$ be as defined in Table 2.3 below. We include the maps ψ_a, ψ_b (from Section 2.3.1) for reference. In the following, the first input to the maps below represents the state of the memory and the second input represents a bit (of information).

Table 2.3: $\psi_a, \psi_{a,0}, \psi_{a,1}, \psi_b, \psi_{b,0}, \psi_{b,1}$ Mappings for two-write code $\mathcal{C}_4$

$\mathbb{Z}_4 \times \mathbb{Z}_2$	ψ_a	$\psi_{a,0}$	$\psi_{a,1}$	ψ_b	$\psi_{b,0}$	$\psi_{b,1}$
(0,0)	0	0	0	1	0	0
(0,1)	1	2	0	0	3	0
(1,0)	2	0	1	1	0	1
(1,1)	1	1	3	3	1	2
(2,0)	2	0	0	2	0	0
(2,1)	3	2	2	3	2	2
(3,0)	0	0	0	0	0	0
(3,1)	3	3	3	3	3	3

We extend the definition of the maps $\psi_{\ell,d}$ (for $\ell = \{a, b\}$, $d = \{0, 1\}$) so that for $\boldsymbol{x}_1 = (x_{1,1}, \ldots, x_{1,n}) \in \mathbb{Z}_4^n$, $\boldsymbol{x}_2 = (x_{2,1}, \ldots, x_{2,n}) \in \mathbb{Z}_2^n$, $\psi_{\ell,d}(\boldsymbol{x}_1, \boldsymbol{x}_2) = (\psi_{\ell,d}(x_{1,1}, x_{2,1}), \ldots, \psi_{\ell,d}(x_{1,n}, x_{2,n}))$.

We will also make use of the maps $\hat{\psi}_0^{-1} : \mathbb{Z}_4 \to \mathbb{Z}_2$ and $\hat{\psi}_1^{-1} : \mathbb{Z}_4 \to \mathbb{Z}_2$ defined in Table 2.4 where $\hat{\psi}_a^{-1}, \hat{\psi}_b^{-1}$ (from Section 2.3.1) are included for completeness.

Table 2.4: $\hat{\psi}_a^{-1}, \hat{\psi}_b^{-1}, \hat{\psi}_0^{-1}, \hat{\psi}_1^{-1}$ Mappings for two-write code $\mathcal{C}_4$

x	$\hat{\psi}_a^{-1}$	$\hat{\psi}_b^{-1}$	$\hat{\psi}_0^{-1}$	$\hat{\psi}_1^{-1}$
0	0	1	0	1
1	1	0	1	0
2	0	0	1	1
3	1	1	1	1

As usual, we extend the definition of the maps $\hat{\psi}_a^{-1}, \hat{\psi}_b^{-1}, \hat{\psi}_0^{-1}, \hat{\psi}_1^{-1}$ so that when the input is a vector the output is the result of applying the map component-wise.

Similar to the maps $\hat{\psi}_a^{-1}, \hat{\psi}_b^{-1}$, the maps $\hat{\psi}_0^{-1}, \hat{\psi}_1^{-1}$ will be used to invert the second argument to $\psi_{a,0}, \psi_{b,0}, \psi_{a,1}, \psi_{b,1}$. This relationship is stated more formally in the following claims.

Claim 4. *Let $\ell \in \{a, b\}$. For any $c \in \mathbb{Z}_4$ and $d \in \mathbb{Z}_2$, where $(c, d) \notin \{(1, 0), (2, 0), (3, 0)\}$, $\psi_{\ell,0}(c, d) \geq c$, $\hat{\psi}_0^{-1}(\psi_{\ell,0}(c, d)) = d$, and $\hat{\psi}_\ell^{-1}(\psi_{\ell,0}(c, d)) = \hat{\psi}_\ell^{-1}(c)$.*

Claim 5. *Let $\ell \in \{a, b\}$. For any $c \in \mathbb{Z}_4$ and $d \in \mathbb{Z}_2$, where $(c, d) \notin \{(0, 0), (2, 0), (3, 0)\}$, $\psi_{\ell,1}(c, d) \geq c$, $\hat{\psi}_1^{-1}(\psi_{\ell,1}(c, d)) = d$, and $\hat{\psi}_\ell^{-1}(\psi_{\ell,1}(c, d)) = \hat{\psi}_\ell^{-1}(c)$.*

Claim 6. *Let $\ell \in \{a, b\}$. For any $c \in \mathbb{Z}_4$ and $d \in \mathbb{Z}_2$ where $(c, d) \notin \{(0, 0), (2, 0), (3, 0)\}$, $\hat{\psi}_0^{-1}(\psi_{\ell,1}(c, d)) = \hat{\psi}_0^{-1}(c)$. Furthermore when $(c, d) \notin \{(1, 0), (2, 0), (3, 0)\}$, $\hat{\psi}_1^{-1}(\psi_{\ell,0}(c, d)) = \hat{\psi}_1^{-1}(c)$.*

We are now ready to state the main result of this section. Recall a symbol-distribution vector $\boldsymbol{p} = (p_0, p_1, p_2, p_3)$ is such that for $0 \leq i \leq 3$, $0 < p_i < 1$, and $\sum_{i=0}^{3} p_i = 1$. We assume in the statement of Theorem 2 that $2^{n(\mathcal{H}(\boldsymbol{p})-\epsilon)}, 2^{n(2p_0+\frac{3p_1}{2}+p_2-\frac{1}{r}-\epsilon)}$ are integer-valued.

Theorem 2. *Let $\boldsymbol{p}$ represent a symbol-distribution vector. For $\epsilon > 0$ and positive integers K, r where $K = \lceil \log_4(r + 1) \rceil + 1$, there exists an $[n + K, 2; 2^{n(\mathcal{H}(\boldsymbol{p})-\epsilon)}, 2^{n(2p_0+\frac{3p_1}{2}+p_2-\frac{1}{r}-\epsilon)}]_4$ Spider Code $\mathcal{C}_4$.*

To prove the result, we define the encoding and decoding maps for the Spider Code $\mathcal{C}_4$. To accomplish this, we will make use of the encoding and decoding maps from an $[n + 1, 2; 2^{n(\mathcal{H}(\boldsymbol{p})-\epsilon)}, 2^{n(1-p_3-\epsilon)}]_4$ code $\tilde{\mathcal{C}}_4$ from Lemma 7. The encoding and decoding maps for $\tilde{\mathcal{C}}_4$ only require a dedicated codeword suffix of size 1, and so we will simply assume that the remaining $K - 1$ symbols in the dedicated codeword suffix for $\mathcal{C}_4$ are initially zero. We describe the code construction in terms of vectors of length n where there are an additional $K - 1$ symbols that can be used to store metadata.

As before, we denote the first generation encoding and decoding maps for $\tilde{\mathcal{C}}_4$ as $\tilde{\mathcal{E}}_{4,1}, \tilde{\mathcal{D}}_{4,1}$ and the second generation encoding and decoding maps for $\tilde{\mathcal{C}}_4$ as $\tilde{\mathcal{E}}_{4,2}, \tilde{\mathcal{D}}_{4,2}$. The first generation encoding and decoding maps for $\mathcal{C}_4$ are identical to $\tilde{\mathcal{E}}_{4,1}, \tilde{\mathcal{D}}_{4,1}$ and will not be further discussed. From Lemma 7, there are $2^{n(\mathcal{H}(\boldsymbol{p})-\epsilon)}$ messages in the first generation where $\epsilon > 0$.

We now turn to defining the encoding map $\mathcal{E}_{4,2}$ for $\mathcal{C}_4$. Let $M_2 = 2^{n(2p_0+\frac{3p_1}{2}+p_2-\frac{1}{r}-\epsilon)}$ where we assume that M_2 is integer-valued and $\boldsymbol{p} = (p_0, p_1, p_2, p_3)$ is a symbol-distribution vector.

Assume that on the first generation, the codeword $\boldsymbol{c}_1 \in \mathcal{C}(\boldsymbol{p}, n)$ (where $\mathcal{C}(\boldsymbol{p}, n)$ is defined in (2.14)) was written and suppose that for the second write we wish to encode the message $m_2 \in \mathbb{Z}_{M_2}$. The input to $\mathcal{E}_{4,2}$ is $(m_2, \boldsymbol{c}_1)$ and the output is $\boldsymbol{c}_2 \in \mathbb{Z}_4^n$.

The encoding map works as follows. In the first step shown in Figure 2.1, we decompose the message m_2 into two messages, $v_{2,1}$ and $v_{2,2}$. This is done so that $v_{2,1}$ is a second generation message of the code $\tilde{\mathcal{C}}_4$. For the code $\tilde{\mathcal{C}}_4$, let $\tilde{M}_2$ be the number of second generation messages so that $\tilde{M}_2 = 2^{n(1-p_3-\epsilon)}$ (see Lemma 7). Using the encoder $\tilde{\mathcal{E}}_{4,2}$ from the code $\tilde{\mathcal{C}}_4$ from Lemma 7 to encode the message $v_{2,1}$, the state of the memory is updated from $\boldsymbol{c}_1$ to $\boldsymbol{z}$ in step 2) using either ψ_a or ψ_b. In step 3), the choice of ψ_a, ψ_b used by the encoder of $\tilde{\mathcal{C}}_4$ is represented as ψ_ℓ. The value of ℓ is stored in the first symbol of the dedicated codeword suffix.

In steps 4) and 5), the function g from (2.16) is used to choose a particular codebook pair from $\mathbf{B}(p_0, p_1, r) = (\mathbf{b_0}, \ldots, \mathbf{b_r})$, which was defined in (2.15). The first codebook chosen from the pair is $\mathcal{C}_2^{(0)}$ and the second codebook chosen is $\mathcal{C}_2^{(1)}$. In step 6) the choice of the codebook pair is stored in the dedicated codeword suffix. At step 7), the message $v_{2,2}$ is decomposed into the messages $v'_{2,2}$ and $v''_{2,2}$ where $v'_{2,2}$ is a second generation message in $\mathcal{C}_2^{(0)}$ and $v''_{2,2}$ is a second generation message in $\mathcal{C}_2^{(1)}$. In steps 8) and 9), the second generation encoder $\mathcal{E}_{2,2}^{(0)}$ for $\mathcal{C}_2^{(0)}$ is used to encode the message $v'_{2,2}$ using the symbols in positions where $\boldsymbol{z}$ has value 0, and similarly the second generation encoder $\mathcal{E}_{2,2}^{(1)}$ for $\mathcal{C}_2^{(1)}$ is used to encode more information using the symbols in positions where $\boldsymbol{z}$ has value 1.

1. Let $\tilde{M}_2 = 2^{n(1-p_3-\epsilon)}$. Let $v_{2,1} \equiv m_2 \mod \tilde{M}_2$ and let $v_{2,2} = \lfloor \frac{m_2}{\tilde{M}_2} \rfloor$.

2. Let $\boldsymbol{z} = \tilde{\mathcal{E}}_{4,2}(v_{2,1}, \boldsymbol{c}_1)$.

3. Suppose ψ_ℓ (where $\ell \in \{a, b\}$) is the map used by $\tilde{\mathcal{E}}_{4,2}$ to encode information as described in Lemma 7. The choice of ℓ is stored in the first symbol of the dedicated codeword suffix.

4. Let $k = g(\frac{|\{i:z_i=0\}|}{n}, p_0, r)$.

5. Let $\mathbf{B}(p_0, p_1, r) = (\mathbf{b}_0, \ldots, \mathbf{b}_r)$, and suppose $(\mathcal{C}_2^{(0)}, \mathcal{C}_2^{(1)}) = \mathbf{b}_k$.

6. Encode $k \in \{0, 1, \ldots, r\}$ with the $K-1$ unused symbols in the dedicated codeword suffix.

7. Let $v'_{2,2} \equiv v_{2,2} \mod M_2^{(k,1)}$ and $v''_{2,2} = \lfloor \frac{v_{2,2}}{M_2^{(k,1)}} \rfloor$. From (2.15), $M_2^{(k,1)}$ refers to the number of second generation messages in $\mathcal{C}_2^{(0)}$.

8. Let $\boldsymbol{u}^{(0)} = \mathcal{E}_{2,2}^{(0)}(v'_{2,2}, \hat{\psi}_0^{-1}(\boldsymbol{z}))$, and $\boldsymbol{u}^{(1)} = \mathcal{E}_{2,2}^{(1)}(v''_{2,2}, \hat{\psi}_1^{-1}(\boldsymbol{z}))$.

9. $\boldsymbol{c}_2 = \psi_{\ell,1}(\psi_{\ell,0}(\boldsymbol{z}, \boldsymbol{u}^{(0)}), \boldsymbol{u}^{(1)})$.

Figure 2.1: Encoding map for $\mathcal{C}_4$

Notice that since the binary pair collection has length $r+1$, $\lceil \log_4(r+1) \rceil$ symbols suffice to store the choice of $(\mathcal{C}_2^{(0)}, \mathcal{C}_2^{(1)})$ in step 6) which is guaranteed in the statement of Theorem 2. In order to maintain the flow of the chapter, we state a number of short claims before moving onto the main results of this section.

Claim 7. *At step 8), $\mathcal{E}_{2,2}^{(0)}$ can encode with rate $\frac{\log_2(M_2^{(k,1)})}{n}$ and $\mathcal{E}_{2,2}^{(1)}$ can encode with rate $\frac{\log_2(M_2^{(k,2)})}{n}$.*

Proof. Recall from Table 2.4 that for $c \in \mathbb{Z}_4$, $\hat{\psi}_0^{-1}(c) = 0$ if and only if $c = 0$ and $\hat{\psi}_1^{-1}(c) = 0$ if and only if $c = 1$. There are at least $(\log_2(M_2^{(k,1)}) + \epsilon n)$ symbols with value 0 in $\boldsymbol{z}$ (as a result of the function g) and there are at least $(\log_2(M_2^{(k,2)}) + \epsilon n)$ symbols with value 1 in

$\boldsymbol{z}$ from Corollary 4. Thus, from Theorem 1 and Lemma 6, $\mathcal{E}_{2,2}^{(0)}$ can encode with a rate of $\frac{\log_2(M_2^{(k,1)})}{n}$ and $\mathcal{E}_{2,2}^{(1)}$ can encode with a rate of $\frac{\log_2(M_2^{(k,2)})}{n}$ at step 8). □

Claim 8. *At step 9) of the encoding, the input to $\psi_{\ell,0}$ is such that there does not exist an index i, $1 \leq i \leq n$, where $(z_i, u_i^{(0)}) \in \{(1,0),(2,0),(3,0)\}$.*

Proof. At step 8), notice that by the definition of the map $\hat{\psi}_0^{-1}$, for $1 \leq i \leq n$ if $z_i \in \{1,2,3\}$, then $\hat{\psi}_0^{-1}(z_i) = 1$. Since $\mathcal{E}_{2,2}^{(0)}$ is the second-generation encoder for a two-write binary WOM-code, if $\hat{\psi}_0^{-1}(z_i) = 1$, then $u_i^{(0)} = 1$ in step 8). Thus, $(z_i, u_i^{(0)}) \notin \{(1,0),(2,0),(3,0)\}$ in step 9). □

For the following claim, let $\psi_{\ell,0}(\boldsymbol{z}, \boldsymbol{u}^{(0)})_i$ denote the i^{th} element of the vector $\psi_{\ell,0}(\boldsymbol{z}, \boldsymbol{u}^{(0)})_i$ at step 9) of the encoding.

Claim 9. *At step 9) of the encoding, the input to $\psi_{\ell,1}$ is such that there does not exist an index i, $1 \leq i \leq n$, where $(\psi_{\ell,0}(\boldsymbol{z}, \boldsymbol{u}^{(0)})_i, u_i^{(1)} \in \{(0,0),(2,0),(3,0)\}$.*

Proof. Let $\psi_{\ell,0}(\boldsymbol{z}, \boldsymbol{u}^{(0)}) = \boldsymbol{z}' = (z_1', \ldots, z_n')$ where $\boldsymbol{z}$ is defined at step 2) and $\boldsymbol{u}^{(0)}$ is defined at step 8). For $1 \leq i \leq n$, if $u_i^{(1)} = 0$ at step 8), we show that z_i' must be 1. Suppose $u_i^{(1)} = 0$. Then, at step 8), since $\mathcal{E}_{2,2}^{(1)}$ is the second generation encoder for a binary WOM-code, if $u_i^{(1)} = 0$ then $\psi_1^{-1}(z_i) = 0$ (since $\boldsymbol{u}$ is the output of the encoding map $\mathcal{E}_{2,2}^{(1)}$ and the cell-state vector given as input to $\mathcal{E}_{2,2}^{(1)}$ is $\psi_1^{-1}(\boldsymbol{z})$). From the definition of the map ψ_1^{-1}, if $\psi_1^{-1}(z_i) = 0$ then $z_i = 1$. In addition, if $z_i = 1$, then $\hat{\psi}_0^{-1}(z_i) = 1$ and since $\mathcal{E}_{2,2}^{(0)}$ is a binary WOM-code $u_i^{(0)} = 1$. Thus, $z_i' = \psi_{\ell,0}(z_i, u_i^{(0)}) = \psi_{\ell,0}(1,1) = 1$ and so the input to $\psi_{\ell,1}$ is not in the set $\{(0,0),(2,0),(3,0)\}$ at step 9). □

We now prove that the WOM-constraint is satisfied.

Lemma 8. *For any first and second generation codewords $\boldsymbol{c}_1, \boldsymbol{c}_2 \in \mathcal{C}_4$, $\boldsymbol{c}_2 \geq \boldsymbol{c}_1$.*

Proof. From Lemma 7, $\boldsymbol{z} \geq \boldsymbol{c}_1$ at step 2) of the encoding map. To show that $\boldsymbol{c}_2 \geq \boldsymbol{c}_1$, it suffices to show that $\boldsymbol{c}_2 \geq \boldsymbol{z}$ in step 9).

From Claim 4 and Claim 8, at step 9) of the encoding map, $\psi_{\ell,0}(\boldsymbol{z}, \boldsymbol{u}^{(0)}) \geq \boldsymbol{z}$ since $(z_i, u_i^{(0)}) \notin \{(1,0),(2,0),(3,0)\}$. Let $\psi_{\ell,0}(\boldsymbol{z}, \boldsymbol{u}^{(0)}) = \boldsymbol{z}' = (z_1', \ldots, z_n')$ where $\boldsymbol{z}$ is defined at step 2) and $\boldsymbol{u}^{(0)}$ is defined at step 8). From Claim 5 and Claim 9, $\psi_{\ell,1}(\boldsymbol{z}', \boldsymbol{u}^{(1)}) \geq \boldsymbol{z}'$ since $(z_i', u_i^{(1)}) \notin \{(0,0),(2,0),(3,0)\}$. Thus, $\psi_{\ell,1}(\psi_{\ell,0}(\boldsymbol{z}, \boldsymbol{u}^{(0)}), \boldsymbol{u}^{(1)}) = \psi_{\ell,1}(\boldsymbol{z}', \boldsymbol{u}^{(1)}) \geq \boldsymbol{z}' = \psi_{\ell,0}(\boldsymbol{z}, \boldsymbol{u}^{(0)})$. Hence, $\boldsymbol{c}_2 = \psi_{\ell,1}(\psi_{\ell,0}(\boldsymbol{z}, \boldsymbol{u}^{(0)}), \boldsymbol{u}^{(1)}) \geq \psi_{\ell,0}(\boldsymbol{z}, \boldsymbol{u}^{(0)}) \geq \boldsymbol{z} \geq \boldsymbol{c}_1$ and the proof is complete. □

The decoder $\mathcal{D}_{4,2}$ is defined as follows. Recall from the encoding, that $\boldsymbol{c}_2 = \mathcal{E}_{4,2}(m_2, \boldsymbol{c}_1)$ is the output of the encoding map for $\mathcal{C}_4$ where m_2 is a message and $\boldsymbol{c}_1$ is the cell-state vector. Recall also that $\tilde{\mathcal{E}}_{4,2}$ and $\tilde{\mathcal{D}}_{4,2}$ are the encoder and decoder for the code $\tilde{\mathcal{C}}_4$. The output of the decoder $\tilde{\mathcal{D}}_{4,2}$ is the codeword $\boldsymbol{c}_2$ and the input is a message $\hat{m}_2 \in \mathbb{Z}_{M_2}$ where $M_2 = 2^{n(2p_0 + \frac{3p_1}{2} + p_2 - \frac{1}{r} - \epsilon)}$.

1. From the dedicated codeword suffix, $\mathcal{D}_{4,2}$ first determines the index k in the binary pair collection so that $(\mathcal{C}_2^{(0)}, \mathcal{C}_2^{(1)}) = \mathbf{b}_k$.
2. Let $\hat{\boldsymbol{u}}^{(0)} = \hat{\psi}_0^{-1}(\boldsymbol{c}_2)$ and $\hat{\boldsymbol{u}}^{(1)} = \hat{\psi}_1^{-1}(\boldsymbol{c}_2)$.
3. Let $\hat{v}_{2,1} = \tilde{\mathcal{D}}_{4,2}(\boldsymbol{c}_2)$, $\hat{v}'_{2,2} = \mathcal{D}_{2,2}^{(0)}(\hat{\boldsymbol{u}}^{(0)})$, and $\hat{v}''_{2,2} = \mathcal{D}_{2,2}^{(1)}(\hat{\boldsymbol{u}}^{(1)})$.
4. Let $\hat{v}_{2,2} = \hat{v}''_{2,2} M_2^{(k,1)} + \hat{v}'_{2,2}$ and $\hat{m}_2 = \hat{v}_{2,2} \tilde{M}_2 + \hat{v}_{2,1}$.

Figure 2.2: Decoding map for $\mathcal{C}_4$

Lemma 9. *If the input to $\mathcal{D}_{4,2}$ is $\boldsymbol{c}_2$ where $\boldsymbol{c}_2 = \mathcal{E}_{4,2}(m_2, \boldsymbol{c}_1)$, then $\hat{m}_2 = m_2$.*

Proof. From step 9) of the encoding map, $\boldsymbol{c}_2 = \psi_{\ell,1}(\psi_{\ell,0}(\boldsymbol{z}, \boldsymbol{u}^{(0)}), \boldsymbol{u}^{(1)})$. From Claim 9, the input to $\psi_{\ell,1}$ is not in the set $\{(0,0),(2,0),(3,0)\}$. Furthermore, from Claim 8 the input to $\psi_{\ell,0}$ is not in the set $\{(1,0),(2,0),(3,0)\}$. Therefore, we can apply the statements from Claim 4, Claim 5, and Claim 6 in the remainder of the proof.

From Claim 5, at step 2) of the decoding, we have $\hat{\boldsymbol{u}}^{(1)} = \hat{\psi}_1^{-1}(\boldsymbol{c}_2) = \hat{\psi}_1^{-1}(\psi_{\ell,1}(\psi_{\ell,0}(\boldsymbol{z}, \boldsymbol{u}^{(0)}), \boldsymbol{u}^{(1)})) = \boldsymbol{u}^{(1)}$. From Claim 6, at step 2) of the decoding, we have $\hat{\boldsymbol{u}}^{(0)} = \hat{\psi}_0^{-1}(\boldsymbol{c}_2) = \hat{\psi}_0^{-1}(\psi_{\ell,1}(\psi_{\ell,0}(\boldsymbol{z}, \boldsymbol{u}^{(0)}), \boldsymbol{u}^{(1)})) =$

$\hat{\psi}_0^{-1}(\psi_{\ell,0}(\boldsymbol{z}, \boldsymbol{u}^{(0)}))$. Then from Claim 4, $\hat{\boldsymbol{u}}^{(0)} = \hat{\psi}_0^{-1}(\boldsymbol{c}_2) = \hat{\psi}_0^{-1}(\psi_{\ell,0}(\boldsymbol{z}, \boldsymbol{u}^{(0)})) = \boldsymbol{u}^{(0)}$.

From Claims 4 and 5 for $\ell \in \{a, b\}$, we have $\hat{\psi}_\ell^{-1}(\boldsymbol{c}_2) = \hat{\psi}_\ell^{-1}(\boldsymbol{z})$ so that from Claim 1, $\tilde{\mathcal{D}}_{4,2}(\boldsymbol{c}_2) = \tilde{\mathcal{D}}_{4,2}(\boldsymbol{z})$. Thus, at step 3) of the decoding procedure, $\hat{v}_{2,1} = v_{2,1}, \hat{v}'_{2,2} = v'_{2,2}$, and $\hat{v}''_{2,2} = v''_{2,2}$ where $v_{2,1}, v'_{2,2}, v''_{2,2}$ are defined in steps 1) and 7) in the encoding procedure. From $v_{2,1}, v'_{2,2}, v''_{2,2}$, the message $\hat{m}_2 = m_2$ and thus the decoding is correct. □

Theorem 2 follows from Lemmas 8 and 9. In the next section, we provide an example of a Spider Code of length 8 that illustrates the decoding maps and the binary pair collection. We summarize the discussion in the following corollary, which gives the maximum achievable rate for a Spider Code of sufficiently long code length. Recall that for a two-write WOM-code $\mathcal{C}_4$, $\mathcal{R}_1(\mathcal{C}_4)$ is the rate of the first generation and $\mathcal{R}_2(\mathcal{C}_4)$ is the rate of the second generation.

Corollary 5. *Suppose $\boldsymbol{p} = (p_0, p_1, p_2, p_3)$ is a symbol-distribution vector. Then, for any $\epsilon > 0$, there exists a WOM-code $\mathcal{C}_4$ where $\mathcal{R}_1(\mathcal{C}_4) = \mathcal{H}(\boldsymbol{p}) - \epsilon$ and $\mathcal{R}_2(\mathcal{C}_4) = 2p_0 + 1.5p_1 + p_2 - \epsilon$.*

Proof. Suppose $\mathcal{C}_4$ is a Spider Code created according to Theorem 2 using the binary pair collection $\mathbf{B}(p_0, p_1, r)$ and the symbol-distribution vector $\boldsymbol{p} = (p_0, p_1, p_2, p_3)$. Let $\epsilon > 0$, $r = \lceil \log_2(n) \rceil$, and $K = \lceil \log_4(r+1) \rceil + 1$. Then the rate of the first generation of $\mathcal{C}_4$ satisfies $\mathcal{R}_1(\mathcal{C}_4) = \frac{n}{n + \lceil \log_4(r+1) \rceil + 1}(\mathcal{H}(\boldsymbol{p}) - \epsilon)$. The rate of the second generation is $\mathcal{R}_2(\mathcal{C}_4) = \frac{n}{n + \lceil \log_4(r+1) \rceil + 1}(2p_0 + 1.5p_1 + p_2 - \frac{1}{r} - \epsilon)$. Thus, when n is large, $\frac{\lceil \log_4(r+1) \rceil}{n} \to 0$, and so $\frac{n}{n + \lceil \log_4(r+1) \rceil + 1} \to 1$. Furthermore, for large n, $\frac{1}{r} \to 0$, giving the result in the corollary. □

From Corollary 5, using a computer search we found that setting $\boldsymbol{p} = (.4070, .2878, .2035, .1017)$ and using the constituent binary codes from [SHP12a] results in the largest possible sum-rate for a Spider Code of 3.2970. This is illustrated as the second row in Table 2.5. In the table below, we also include the rates achievable using existing non-binary code constructions as reported in [GYDSVW11] and [YS12]. Since the authors of [YS12] did not provide the optimal rates achievable using their construction (except when $q = 8, 16$), the table below reports the rates achieved using the constructions from [GYDSVW11] for $q = 4, 32, 64$ with capacity-achieving binary code constituents. As can be seen below, our construction improves upon

the previous best sum-rates reported in [GYDSVW11] and [YS12] for sufficiently long code lengths.

Table 2.5: Achievable sum-rates for two-writes using Spider Codes

q	New Sum-Rate	Sum-Rate From [GYDSVW11], [YS12]	Capacity
4	3.2970	3.1700([GYDSVW11])	3.3219
8	4.8820	4.8290 ([YS12])	5.1699
16	6.5940	6.5585 ([YS12])	7.0875
32	8.1790	7.9250 ([GYDSVW11])	9.0444
64	10.6979	9.5100 ([GYDSVW11])	11.0224

The third row in Table 2.5 is the result of using a code constructed according to Corollary 2 with two constituent codes. The first constituent code is a Spider code with rate 3.2970 and the second constituent is a binary capacity-achieving code (from [SHP12a]) which has sum-rate 1.5850. The rate of the resulting code is the sum of the rates of the constituent codes so that the resulting code has a sum-rate of 4.8820. The fourth row in Table 2.5 is the result of using the Expansion Construction with two Spider Codes as constituents. The fifth row in Table 2.5 is the result of using the code listed in the fourth row of the table along with a binary capacity-achieving code as constituents according to Corollary 2. The final row of the table is the result of using the Ladder Construction with $L = 13$ and a Spider code as a constituent.

We remark that the codes from [SHP12b] have rates that approach capacity. A potential drawback of these codes is that if polynomial time encoding/decoding procedures are desired, then the resulting codes are necessarily long. Few high rate, finite length codes have been identified and most of the work appears to be limited to the binary case as in [YKSVW12]. In this section, we presented the Spider Codes using the binary codes from [SHP12a], where the codes from [SHP12a] also require long code lengths, to illustrate the best achievable rates. However, we note that our construction has the advantage that it requires only binary constituent codebooks, so that finding high-rate short length binary codes (that satisfy Lemma 6) has an additional benefit of resulting in high-rate short length non-binary codes.

Furthermore, the ideas used in our Spider Code construction can be used for creating codes of reasonable block lengths as demonstrated in the next subsection.

2.3.3 A Finite Length Spider Code

We now illustrate how the ideas from the Spider Code construction can be used to construct a code of short code length. To make things simpler, we chose a code length of 8. The constituent binary codes described in the example below can be constructed using the coset-coding technique described in [CGM86]. We will use the following binary codebooks:

1. $[7, 2; 8, 2^6]_2$ codebook $\mathcal{C}_2$
2. $[7, 2; 127, 2]_2$ codebook $\mathcal{C}_2'$
3. $[7, 2; 64, 2^3]_2$ codebook $\mathcal{C}_2''$
4. $[7, 2; 2^7, 1]_2$ codebook $\mathcal{C}_2'''$.

Suppose w is a positive integer. Let $\mathcal{C}_{wt(w)}$ denote the set of vectors from $\mathbb{Z}_4^n$ whose weight is at most w. For a two-write WOM-code $\mathcal{C}$ over $\mathbb{Z}_4^n$, let $G_1(\mathcal{C})$ denote the set of first generation codewords for $\mathcal{C}$. Then, $G_1(\mathcal{C}_2) = \mathcal{C}_{wt(1)}$, $G_1(\mathcal{C}_2') = \mathcal{C}_{wt(6)}$, $G_1(\mathcal{C}_2'') = \mathcal{C}_{wt(3)}$, and $G_1(\mathcal{C}_2''') = \mathcal{C}_{wt(7)}$.

We note that, unlike in the description of the encoding map in Figure 2.1, in the example below, we use only a single symbol to store the dedicated codeword suffix. As will be described shortly, this is because the binary pair collection contains only two pairs of codebooks and the choice of ℓ requires only a single bit of information. Furthermore, the selection of the binary codebook pair will differ slightly from the encoding map in Figure 2.1. These modifications are necessary because we are constructing a finite length Spider Code whereas the description in Section 2.3 was appropriate for longer block lengths.

Example 1. *Suppose we wish to construct a two-write Spider Code of length* 8*. We will use the first* 7 *symbols to encode information and the remaining symbol will be used as*

the dedicated codeword suffix. For the first write we store vectors from $\mathcal{C}(\boldsymbol{p}, 7)$ *where* $\boldsymbol{p} = (\frac{2}{7}, \frac{3}{7}, \frac{1}{7}, \frac{1}{7})$. *Suppose, on the first write, we store the vector* $(2, 1, 0, 1, 0, 1, 3, \mathbf{0})$, *where the dedicated codeword suffix is the last symbol (in bold) and it has value* 0. *We refer to the vector* $(2, 1, 0, 1, 0, 1, 3)$ *as* $\boldsymbol{c}_1 = (c_{1,1}, \ldots, c_{1,7})$.

For the second write, we make use of the $[7, 2; 8, 2^6]_2$ *codebook* $\mathcal{C}_2$ *in order to determine the vector* $\boldsymbol{z}$ *described at step 2) of the encoding algorithm shown in Figure 2.1. We first determine the cell-state vector* $\boldsymbol{z}' = (z'_1, \ldots, z'_7)$ *which will be used as input to the encoder for* $\mathcal{C}_2$. *Recall from Lemma 7, that for* $1 \leq i \leq 7$, $z'_i = 1$ *if and only if* $c_{1,i} = 3$ *which implies* $\boldsymbol{z}' = (0, 0, 0, 0, 0, 0, 1)$. *Notice that since the weight of* $\boldsymbol{z}'$ *is* 1, $\boldsymbol{z}'$ *is a first generation codeword in* $\mathcal{C}_2$. *Suppose the second generation codeword* $\boldsymbol{u} = (1, 1, 0, 0, 0, 1, 1) \in \mathcal{C}_2$ *is chosen. Since* $\boldsymbol{u}$ *is a second generation codeword in* $\boldsymbol{u}$ *(and we assume that* $\boldsymbol{z}'$ *is the cell-state vector), notice that* $\boldsymbol{u} \geq \boldsymbol{z}'$. *Since* $|\{i : c_{1,i} = 1, u_i = 1\}| \geq \{i : c_{1,i} = 1, u_i = 0\}|$, *we choose* $\psi_\ell = \psi_a$. *Then the state of the memory is updated to*

$$\boldsymbol{z} = \psi_a(\boldsymbol{c}_1, \boldsymbol{u}) = (3, 1, 0, 2, 0, 1, 3)$$

in step 2).

In the following, we perform the equivalent of steps 4), 5), 6), and 7) from Figure 2.1. We will make use of the binary pair collection $(([7, 2; 2^7, 1]_2, [7, 2; 64, 2^2]_2), ([7, 2; 127, 2]_2, [7, 2; 127, 2]_2))$. *The codebook* $[7, 2; 64, 2^2]_2$ *can be constructed by removing second generation messages from the codebook* $[7, 2; 64, 2^3]_2$. *If all the symbols with a value* 0 *in* $\boldsymbol{c}_1$ *were incremented to a value* 1 *in* $\boldsymbol{z}$ *then we would set* $(\mathcal{C}^{(0)}, \mathcal{C}^{(1)}) = ([7, 2; 2^7, 1]_2, [7, 2; 64, 2^2]_2)$ *at step 5) of the encoding map. Otherwise, we set* $(\mathcal{C}^{(0)}, \mathcal{C}^{(1)}) = ([7, 2; 127, 2]_2, [7, 2; 127, 2]_2)$. *Notice that in either case the number of total second generation messages (the product of the second generation messages in* $\mathcal{C}^{(0)}$ *and the second generation messages in* $\mathcal{C}^{(1)}$*) is the same. If we choose the* i*-th entry from the binary pair collection (where* $1 \leq i \leq 2$*) and* $\ell = a$ *then we store the symbol* $i - 1$ *in the dedicated codeword suffix. If* $\ell = b$, *then we store the symbol* $i + 1$ *in*

the dedicated codeword suffix. For our example, since there are 2 *symbols with value* 0 *in both* $\boldsymbol{z}$ *and* $\boldsymbol{c}_1$*, we set* $(\mathcal{C}^{(0)}, \mathcal{C}^{(1)}) = ([7, 2; 127, 2]_2, [7, 2; 127, 2]_2)$*. We store the symbol* 1 *in the dedicated codeword suffix.*

Notice that $\hat{\psi}_0^{-1}(\boldsymbol{z}) = (1, 1, 0, 1, 0, 1, 1)$ *and* $\hat{\psi}_1^{-1}(\boldsymbol{z}) = (1, 0, 1, 1, 1, 0, 1)$*. At step 8), we proceed by choosing a second generation codeword* $\boldsymbol{u}^{(0)}$ *from* $\mathcal{C}^{(0)}$ *provided the cell-state vector* $\hat{\psi}_0^{-1}(\boldsymbol{z})$*. Similarly we choose a second generation codeword* $\boldsymbol{u}^{(1)}$ *from* $\mathcal{C}^{(1)}$ *provided the cell-state vector* $\hat{\psi}_1^{-1}(\boldsymbol{z})$*. Suppose* $\boldsymbol{u}^{(0)} = (1, 1, 1, 1, 0, 1, 1)$ *and* $\boldsymbol{u}^{(1)} = (1, 0, 1, 1, 1, 1, 1)$*. Then, at step 9) we have* $\boldsymbol{c}_2 = \psi_{a,1}(\psi_{a,0}(\boldsymbol{z}, \boldsymbol{u}^{(0)}), \boldsymbol{u}^{(1)}) = (3, 1, 2, 2, 0, 3, 3)$.

Since $|\mathcal{C}(\boldsymbol{p}, n)| = 420$*, there are 420 first generation codewords in the resulting Spider Code. In the second generation, regardless of the choice of* $(\mathcal{C}^{(0)}, \mathcal{C}^{(1)})$*, we store an additional* $2^6 \cdot 2^2$ *messages so that there are* 256 *messages in the second generation.*

In the next section, we consider the application of our non-binary code constructions to the problem of constructing fixed-rate codes.

2.4 Fixed-rate WOM-codes

In this section, we study non-binary fixed-rate codes. Most of the work in this area has focused solely on the binary case. In [RS82], the capacity of a two-write fixed-rate binary WOM-code was given. In [HEE85], this result was extended and a recursive expression for the fixed-rate capacity for t-write binary WOM-codes was derived.

We begin the section by deriving expressions for the fixed-rate capacity of t-write q-ary WOM-codes. Using these expressions, we then provide lower bounds on the capacity of t-write non-binary fixed-rate WOM-codes for $t \leq 4$ and $q \leq 8$. Afterwards, we use the constructions introduced earlier in the chapter to produce fixed-rate codes, and these codes are then compared against the bounds in this section.

Recall from Definition 3 that $\boldsymbol{p}_q = (p_0, \ldots, p_{q-1})$ is a symbol-distribution vector if for $0 \leq i \leq q-1$, $0 < p_i < 1$ and $\sum_{i=0}^{q-1} p_i = 1$. Unlike the previous section, we do not assume

the symbol-distribution vectors in this section have a length of 4.

2.4.1 Upper and Lower Bounds on Fixed-Rate Capacity

We begin by considering a result that follows from [FV99], [GD11]. Recall, that for a 2-write WOM-code $\mathcal{C}_q$, $\mathcal{R}_1(\mathcal{C}_q)$ refers to the rate of the first generation and $\mathcal{R}_2(\mathcal{C}_q)$ refers to the rate of the second generation.

Similar to the notation in the previous section, for a symbol distribution vector $\boldsymbol{p}_q$ and an integer n, let

$$\mathcal{C}(\boldsymbol{p}_q, n) = \{\boldsymbol{x} =(x_1, \ldots, x_n) \in \mathbb{Z}_q^n : \\ 0 \leq a \leq q-1, |\{i : x_i = a\}| = p_a n\},$$

where we assume that $p_i n$ is integer-valued for $0 \leq i \leq q-1$.

In the following lemma, suppose q is a positive integer and $\boldsymbol{p}_q^{(1)}, \boldsymbol{p}_q^{(2)}, \boldsymbol{p}_{q-1}^{(2)}, \ldots \boldsymbol{p}_2^{(2)}$ are symbol-distribution vectors. Under this notation, the superscripts refer to the generation. Let $\boldsymbol{p}_q^{(1)} = (p_0, \ldots, p_{q-1})$.

Lemma 10. *(cf. [FV99], Lemma 3.1) For any symbol distribution vectors $\boldsymbol{p}_q^{(1)}, \boldsymbol{p}_q^{(2)}, \boldsymbol{p}_{q-1}^{(2)}, \ldots \boldsymbol{p}_2^{(2)}$, there exists a q-ary 2-write WOM-code $\mathcal{C}_q$ where*

$$\mathcal{R}_1(\mathcal{C}_q) \leq \mathcal{H}(\boldsymbol{p}_q^{(1)}),$$
$$\mathcal{R}_2(\mathcal{C}_q) \leq p_0 \mathcal{H}(\boldsymbol{p}_q^{(2)}) + p_1 \mathcal{H}(\boldsymbol{p}_{q-1}^{(2)}) \ldots + p_{q-2} \mathcal{H}(\boldsymbol{p}_2^{(2)}).$$

It was also shown in [FV99] that for any q-ary 2-write WOM-code $\mathcal{C}_q$ there exists $\boldsymbol{p}_q^{(1)}, \boldsymbol{p}_q^{(2)}, \boldsymbol{p}_{q-1}^{(2)}, \ldots \boldsymbol{p}_2^{(2)}$ where Lemma 10 holds. Thus, the fixed-rate capacity for a two-write WOM-code over $\mathbb{Z}_q$ is $2\mathcal{R}$ where $2\mathcal{R}$ is given by

$$\max_{\boldsymbol{p}_q^{(1)},\boldsymbol{p}_q^{(2)},\ldots,\boldsymbol{p}_2^{(2)}} \{2\mathcal{R} : \mathcal{R} \leq \mathcal{H}(\boldsymbol{p}_q^{(1)}),$$
$$\mathcal{R} \leq p_0\mathcal{H}(\boldsymbol{p}_q^{(2)}) + \ldots + p_{q-2}\mathcal{H}(\boldsymbol{p}_2^{(2)}), \}, \tag{2.17}$$

where $\boldsymbol{p}_q^{(1)}, \boldsymbol{p}_q^{(2)}, \ldots, \boldsymbol{p}_2^{(2)}$ are symbol-distribution vectors.

The following lemma provides an expression for the fixed-rate capacity for a two-write WOM-code that involves fewer variables than (2.17) and is therefore more computationally tractable. In particular, notice that the variables $\boldsymbol{p}_q^{(2)}, \boldsymbol{p}_{q-1}^{(2)}, \ldots \boldsymbol{p}_2^{(2)}$ are no longer necessary.

Lemma 11. *For an integer $q > 2$, the fixed-rate capacity for a two-write WOM-code over $\mathbb{Z}_q$ is $2 \cdot \mathcal{H}(\boldsymbol{p}_q^{(u)})$ where*

$$\boldsymbol{p}_q^{(u)} = \arg\max_{\boldsymbol{p}_q^{(1)}} \left\{ \min \left(\mathcal{H}(\boldsymbol{p}_q^{(1)}), \sum_{i=0}^{q-2} p_i \log_2(q-i) \right) \right\} \tag{2.18}$$

and $\boldsymbol{p}_q^{(1)} = (p_0, \ldots, p_{q-1})$ is a symbol-distribution vector.

Proof. We first note that since $\mathcal{H}(\boldsymbol{p}_q^{(1)})$ and $\sum_{i=0}^{q-2} p_i \log_2(q-i)$ are concave functions over $\boldsymbol{p}_q^{(1)}$, it follows that $\min\left(\mathcal{H}(\boldsymbol{p}_q^{(1)}), \sum_{i=0}^{q-2} p_i \log_2(q-i)\right)$ is also concave over $\boldsymbol{p}_q^{(1)}$ (*cf.* [BV04]). Since the set of feasible points for $p_0, p_1, \ldots, p_{q-1}$ is concave, it follows from [BV04] that there exists a unique solution to (2.18). Let $\boldsymbol{p}_q^{(u)} = (p_0^{(u)}, \ldots, p_{q-1}^{(u)})$ be as in (2.18) and suppose $\mathcal{R}^{(u)} = \min\left(\mathcal{H}(\boldsymbol{p}_q^{(u)}), \sum_{i=0}^{q-2} p_i^{(u)} \log_2(q-i)\right)$.

Suppose, on the contrary, that there exists a fixed-rate two-write WOM-code $\mathcal{C}_q'$ that has a rate $\mathcal{R}'$ on each generation where $\mathcal{R}' > \mathcal{R}^{(u)}$. From Lemma 10, there exist symbol-distribution vectors $\boldsymbol{p}_q'^{(1)} = (p_0', \ldots, p_{q-1}')$, $\boldsymbol{p}_q'^{(2)}$, $\boldsymbol{p}_{q-1}'^{(2)}$, …, $\boldsymbol{p}_2'^{(2)}$, where $\mathcal{R}' \leq \mathcal{H}(\boldsymbol{p}_q'^{(1)})$ and $\mathcal{R}' \leq p_0'\mathcal{H}(\boldsymbol{p}_q'^{(2)}) + \ldots + p_{q-2}'\mathcal{H}(\boldsymbol{p}_2'^{(2)})$. Furthermore, from the concavity of the log function, $\mathcal{R}' \leq p_0' \log_2(q) + \ldots + p_{q-2}' \log_2(2)$. However, then $\min\left(\mathcal{H}(\boldsymbol{p}_q'^{(1)}), \sum_{i=0}^{q-2} p_i' \log_2(q-i)\right) \geq \mathcal{R}' > \mathcal{R}^* = \min\left(\mathcal{H}(\boldsymbol{p}_q^*), \sum_{i=0}^{q-2} p_i^* \log_2(q-i)\right)$, and we arrive at a contradiction. □

For $q \leq 8$, solutions to (2.18) were found using a computer search. Our results are in Table 2.6. The values for the symbol-distribution vectors are provided in columns 4-11. The second row in the table is from [RS82] and it is included for completeness. We now produce a closed-form upper bound for the fixed-rate capacity of a two-write non-binary WOM-code.

Lemma 12. *The fixed-rate capacity of a two-write WOM-code over $\mathbb{Z}_q$ is at most $\frac{2}{3}\log_2\left(\frac{q(q+1)(2q+1)}{6}\right)$.*

Proof. Let $2\mathcal{R}^{(u)}$ denote the fixed-rate capacity of a two-write WOM-code over $\mathbb{Z}_q$. From (2.17) for some symbol-distribution vector $\boldsymbol{p}_q^{(1)} = (p_0, p_1, \ldots, p_{q-1})$, we can write

$$2\mathcal{R}^{(u)} \leq \mathcal{H}(\boldsymbol{p}_q^{(1)}) + \sum_{i=0}^{q-1} p_i \log_2(q-i).$$

Since the rates of the first and second generations are equal then $\mathcal{R}^{(u)} \leq \sum_{i=0}^{q-1} p_i \log_2(q-i)$ and so

$$\begin{aligned} 3\mathcal{R}^{(u)} &\leq \mathcal{H}(\boldsymbol{p}_q^{(1)}) + 2\sum_{i=0}^{q-1} p_i \log_2(q-i) \\ &= \mathcal{H}(\boldsymbol{p}_q^{(1)}) + \sum_{i=0}^{q-1} p_i \log_2((q-i)^2) \\ &= \sum_{i=0}^{q-1} p_i \log_2\left(\frac{(q-i)^2}{p_i}\right). \end{aligned}$$

By the concavity of the log function, we have

$$\begin{aligned} 3\mathcal{R}^{(u)} &\leq \log_2\left(\sum_{i=0}^{q-1}(q-i)^2\right) \\ &= \log_2\left(\frac{q(q+1)(2q+1)}{6}\right). \end{aligned}$$

Thus, $\mathcal{R}^{(u)} \leq \frac{1}{3}\log_2\left(\frac{q(q+1)(2q+1)}{6}\right)$ and $2\mathcal{R}^{(u)} \leq \frac{2}{3}\log_2\left(\frac{q(q+1)(2q+1)}{6}\right)$. □

The numbers in the third column of Table 2.6 are the upper bound from Lemma 12. For larger values of q, our upper bound can be evaluated using Lemma 13 below.

Lemma 13. *The fixed-rate capacity of a two-write WOM-code over $\mathbb{Z}_q$ is at least* $2 \cdot \sum_{i=2}^{q} \frac{i}{q(q+1)/2} \log_2(i)$.

Proof. We first show that for any two-write q-ary capacity-achieving WOM-code $\mathcal{C}_q$, $\mathcal{R}_1(\mathcal{C}_q) \geq \mathcal{R}_2(\mathcal{C}_q)$. From [FV99] and Lemma 10, we have that $\mathcal{R}_1(\mathcal{C}_q) = \mathcal{H}(\boldsymbol{p}_q^{(1)})$ where $\boldsymbol{p}_q^{(1)} = (p_0, p_1, \ldots, p_{q-1}) = (\frac{q}{q(q+1)/2}, \frac{q-1}{q(q+1)/2}, \ldots, \frac{1}{q(q+1)/2})$ and $\mathcal{R}_2(\mathcal{C}_q) = \sum_{i=1}^{q} \frac{i}{q(q+1)/2} \log_2(i)$. In this case, we have

$$\begin{aligned}
\mathcal{R}_2(\mathcal{C}_q) - \mathcal{R}_1(\mathcal{C}_q) &= \sum_{i=1}^{q} \frac{i}{q(q+1)/2} \log_2(i) + \\
&\quad \sum_{i=1}^{q} \frac{i}{q(q+1)/2} \log_2\left(\frac{i}{q(q+1)/2}\right) \\
&= \sum_{i=1}^{q} \frac{i}{q(q+1)/2} \log_2\left(\frac{i^2}{q(q+1)/2}\right) \\
&\leq \log_2\left(\sum_{i=1}^{q} \frac{i^3}{q^2(q+1)^2/4}\right) \\
&= \log_2(1) = 0.
\end{aligned}$$

Since we have just shown $\mathcal{R}_2(\mathcal{C}_q) \leq \mathcal{R}_1(\mathcal{C}_q)$, the statement in the lemma holds since $2\mathcal{R}_2(\mathcal{C}_q) = 2 \cdot \sum_{i=2}^{q} \frac{i}{q(q+1)/2} \log_2(i)$. □

Table 2.6: Fixed-Rate Capacity for two-write WOM-codes

q	Capacity	U.B. Lemma 12	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7
2	1.55	1.55	0.773	0.227	-	-	-	-	-	-
3	2.54	2.54	0.610	0.300	0.090	-	-	-	-	-
4	3.27	3.27	0.501	0.304	0.15	0.045	-	-	-	-
5	3.85	3.85	0.424	0.288	0.175	0.087	0.026	-	-	-
6	4.33	4.34	0.367	0.268	0.182	0.111	0.055	0.0170	-	-
7	4.75	4.75	0.324	0.248	0.181	0.123	0.075	0.037	0.012	-
8	5.11	5.12	0.290	0.230	0.176	0.129	0.088	0.053	0.026	0.008

We now extend the ideas from the two-write case to multiple writes. We begin with a recursive expression that follows from [FV99] for the set of possible rates for a t'-write WOM-code $\mathcal{C}_{q'}$. For all j where $1 \leq j \leq t'$, let $\mathcal{R}_j(\mathcal{C}_{q'})$ denote the rate of $\mathcal{C}_{q'}$ on the j^{th} write.

We say that a rate vector $(\mathcal{R}^{(1)}_{t',q'}, \mathcal{R}^{(2)}_{t',q'}, \ldots, \mathcal{R}^{(t')}_{t',q'})$ is achievable if there exists a t'-write q'-ary WOM-code $\mathcal{C}_{q'}$ where $\mathcal{R}_j(\mathcal{C}_{q'}) = \mathcal{R}^{(j)}_{t',q'}$ for all j where $1 \leq j \leq t'$.

Lemma 14. *(cf. [FV99], Lemma 3.1) For any symbol distribution vector $\boldsymbol{p}^{(1)}_q$, and any achievable rate vectors $(\mathcal{R}^{(1)}_{t-1,q}, \ldots, \mathcal{R}^{(t-1)}_{t-1,q}), (\mathcal{R}^{(1)}_{t-1,q-1}, \ldots, \mathcal{R}^{(t-1)}_{t-1,q-1}) \ldots, (\mathcal{R}^{(1)}_{t-1,2}, \ldots, \mathcal{R}^{(t-1)}_{t-1,2})$, there exists a q-ary t-write WOM-code $\mathcal{C}_q$ where*

$$\begin{aligned}
\mathcal{R}_1(\mathcal{C}_q) &\leq \mathcal{H}(\boldsymbol{p}^{(1)}_q), \\
\mathcal{R}_2(\mathcal{C}_q) &\leq p_0\mathcal{R}^{(1)}_{t-1,q} + p_1\mathcal{R}^{(1)}_{t-1,q-1} + \ldots + p_{q-2}\mathcal{R}^{(1)}_{t-1,2}, \\
\mathcal{R}_3(\mathcal{C}_q) &\leq p_0\mathcal{R}^{(2)}_{t-1,q} + p_1\mathcal{R}^{(2)}_{t-1,q-1} + \ldots + p_{q-2}\mathcal{R}^{(2)}_{t-1,2}, \\
&\vdots \\
\mathcal{R}_t(\mathcal{C}_q) &\leq p_0\mathcal{R}^{(t-1)}_{t-1,q} + p_1\mathcal{R}^{(t-1)}_{t-1,q-1} + \ldots + p_{q-2}\mathcal{R}^{(t-1)}_{t-1,2}.
\end{aligned}$$

Similar to before, it was also shown in [FV99] that for any q-ary t-write WOM-code $\mathcal{C}_q$, there exists a symbol distribution vector $\boldsymbol{p}^{(1)}_q$ and achievable rate vectors $(\mathcal{R}^{(1)}_{t-1,q}, \ldots, \mathcal{R}^{(t-1)}_{t-1,q})$, $(\mathcal{R}^{(1)}_{t-1,q-1}, \ldots, \mathcal{R}^{(t-1)}_{t-1,q-1}), \ldots, (\mathcal{R}^{(1)}_{t-1,2}, \ldots, \mathcal{R}^{(t-1)}_{t-1,2})$ where Lemma 14 holds. Thus, the fixed-rate capacity for a t-write WOM-code is given by

$$\begin{aligned}
\max\{t\mathcal{R} : &\mathcal{R} \leq \mathcal{H}(\boldsymbol{p}^{(1)}_q), \\
&\mathcal{R} \leq p_0\mathcal{R}^{(1)}_{t-1,q} + \ldots + p_{q-2}\mathcal{R}^{(1)}_{t-1,2}, \\
&\vdots \\
&\mathcal{R} \leq p_0\mathcal{R}^{(t-1)}_{t-1,q} + \ldots + p_{q-2}\mathcal{R}^{(t-1)}_{t-1,2}\},
\end{aligned} \tag{2.19}$$

where the max function is taken over all possible choices for $\boldsymbol{p}^{(1)}_q$ and achievable rate-vectors $(\mathcal{R}^{(1)}_{t-1,q}, \ldots, \mathcal{R}^{(t-1)}_{t-1,q})$, $(\mathcal{R}^{(1)}_{t-1,q-1}, \ldots, \mathcal{R}^{(t-1)}_{t-1,q-1})$, $\ldots$, $(\mathcal{R}^{(1)}_{t-1,2}, \ldots, \mathcal{R}^{(t-1)}_{t-1,2})$.

In the following lemma, suppose q, t are positive integers where $q > 2$ and $t \geq 2$ and $\boldsymbol{p}^{(1)}_q = (p_0, \ldots, p_{q-1})$ is a symbol-distribution vector. The result is immediate from (2.19).

Lemma 15. *The fixed-rate capacity of a t-write WOM-code over $\mathbb{Z}_q$ is*

$$\max\left\{\min\left(\mathcal{H}(\boldsymbol{p}_q^{(1)}), \sum_{i=0}^{q-2} p_i \mathcal{R}_{t-1,q-i}^{(1)}, \ldots, \sum_{i=0}^{q-2} p_i \mathcal{R}_{t-1,q-i}^{(t-1)}\right)\right\}, \tag{2.20}$$

where the max *function is taken over all possible choices for* $\boldsymbol{p}_q^{(1)}, (\mathcal{R}_{t-1,q}^{(1)}, \ldots, \mathcal{R}_{t-1,q}^{(t-1)})$, $(\mathcal{R}_{t-1,q-1}^{(1)}, \ldots, \mathcal{R}_{t-1,q-1}^{(t-1)}), \ldots, (\mathcal{R}_{t-1,2}^{(1)}, \ldots, \mathcal{R}_{t-1,2}^{(t-1)})$.

The following lower bound on the capacity follows from Lemma 15. We assume that, as in Lemma 15, $(\mathcal{R}_{t-1,q}^{(1)}, \ldots, \mathcal{R}_{t-1,q}^{(t-1)})$, $(\mathcal{R}_{t-1,q-1}^{(1)}, \ldots, \mathcal{R}_{t-1,q-1}^{(t-1)}), \ldots, (\mathcal{R}_{t-1,2}^{(1)}, \ldots, \mathcal{R}_{t-1,2}^{(t-1)})$ are achievable rate vectors and $\boldsymbol{p}_q^{(1)} = (p_0, \ldots, p_{q-1})$ is a symbol-distribution vector.

Corollary 6. *If for all i where* $0 \leq i \leq q-2$, $\mathcal{R}_{t-1,q-i}^{(1)} = \mathcal{R}_{t-1,q-i}^{(2)} \cdots = \mathcal{R}_{t-1,q-i}^{(t-1)}$, *then there exists a t-write WOM-code over* $\mathbb{Z}_q$ *with sum-rate* $t \cdot \mathcal{H}(\boldsymbol{p}_q)$, *where*

$$\boldsymbol{p}_q = \arg\max_{\boldsymbol{p}_q^{(1)}} \left\{\min\left(\mathcal{H}(\boldsymbol{p}_q^{(1)}), \sum_{i=0}^{q-2} p_i \mathcal{R}_{t-1,q-i}^{(1)}\right)\right\}.$$

Using Corollary 6, we were able to compute lower bounds on the fixed-rate capacity for $t = 2, 3, 4$ when $q \leq 8$. Our results are summarized in Table 2.7.

Table 2.7: Lower Bound on Fixed-Rate Capacity for 2,3, and 4-write WOM-codes

q	2-write	3-write	4-write
3	2.54	3.24	3.80
4	3.27	4.23	5.01
5	3.85	5.03	6.00
6	4.33	5.70	6.84
7	4.75	6.29	7.58
8	5.11	6.80	8.23

The following lemma is analogous to Lemma 12. We assume that, as in Lemma 15, $(\mathcal{R}_{t-1,q}^{(1)}, \ldots, \mathcal{R}_{t-1,q}^{(t-1)})$, $(\mathcal{R}_{t-1,q-1}^{(1)}, \ldots, \mathcal{R}_{t-1,q-1}^{(t-1)}) \ldots, (\mathcal{R}_{t-1,2}^{(1)}, \ldots, \mathcal{R}_{t-1,2}^{(t-1)})$ are achievable rate vectors where for $0 \leq i \leq q-2$, $\mathcal{R}_{t-1,q-i}^{(1)} = \mathcal{R}_{t-1,q-i}^{(2)} \cdots = \mathcal{R}_{t-1,q-i}^{(t-1)}$, and $\boldsymbol{p}_q^{(1)} = (p_0, \ldots, p_{q-1})$ is a symbol-distribution vector.

Lemma 16. *A fixed-rate t-write WOM-code over $\mathbb{Z}_q$ created according to Corollary 6 has rate at most $\frac{t}{t+1}\log_2\left(\sum_{i=0}^{q-1} 2^{t\mathcal{R}^{(1)}_{t-1,q-i}}\right)$.*

Proof. Let $t\mathcal{R}^*_{t,q}$ denote the maximum rate of a t-write WOM-code created according to Corollary 6. Then, from (2.19) and Corollary 6, for some symbol-distribution vector $\boldsymbol{p}^{(1)}_q = (p_0, p_1, \ldots, p_{q-1})$, we can write

$$t\mathcal{R}^*_{t,q} \leq \mathcal{H}(\boldsymbol{p}^{(1)}_q) + \sum_{i=0}^{q-1} p_i \log_2\left(2^{(t-1)\mathcal{R}^{(1)}_{t-1,q-i}}\right).$$

Following the proof of Lemma 12, $\mathcal{R}^*_{t,q} \leq \sum_{i=0}^{q-1} p_i \log_2\left(2^{\mathcal{R}^{(1)}_{t-1,q-i}}\right)$ and so

$$\begin{aligned}
(t+1)\mathcal{R}^*_{t,q} &\leq \mathcal{H}(\boldsymbol{p}_q) + \sum_{i=0}^{q-1} p_i \log_2\left(2^{(t-1)\mathcal{R}^{(1)}_{t-1,q-i}}\right) \\
&\quad + \sum_{i=0}^{q-1} p_i \log_2\left(2^{\mathcal{R}^{(1)}_{t-1,q-i}}\right) \\
&= \mathcal{H}(\boldsymbol{p}_q) + \sum_{i=0}^{q-1} p_i \log_2\left(2^{t\mathcal{R}^{(1)}_{t-1,q-i}}\right) \\
&\leq \log_2\left(\sum_{i=0}^{q-1} 2^{t\mathcal{R}^{(1)}_{t-1,q-i}}\right),
\end{aligned}$$

where the last inequality follows from the concavity of the log function. Therefore, $t\mathcal{R}^*_{t,q} \leq \frac{t}{t+1}\log_2\left(\sum_{i=0}^{q-1} 2^{t\mathcal{R}^{(1)}_{t-1,q-i}}\right)$ and the proof is complete.

□

In the following subsection, the results of using the constructions in this work to produce fixed-rate codes are presented and compared against the bounds from this subsection.

2.4.2 Achieved Fixed-Rates

In this subsection, we report on the results of using the Expansion Construction (from Section 2.2.1), the Ladder Construction (from Section 2.2.2), and the Spider Codes from Section 2.3 to produce fixed-rate two-write codes.

Table 2.8: Fixed-Rates achieved for two-writes

q	U.B. on Capacity	Expansion	Ladder	S.C/Expansion
4	3.27	2.91	-	3.24
8	5.11	4.36	4.62	4.79
16	7.03	5.82	-	6.48
32	8.99	7.27	8.37	8.03

The second column of Table 2.8 is an upper bound on the fixed-rate capacity for the given value of q. The codes under the column for the Expansion (Construction) are the result of using the fixed-rate binary code from [YKSVW12] of rate 1.4546.

The code listed in the third row under the Ladder column is the result of using the Ladder Construction with $L = 3$ with the fixed-rate binary code from [YKSVW12] of rate 1.4546. The code listed in the fifth row of the Ladder column is the result of using the Ladder Construction with $L = 11$ and the same fixed-rate binary code from [YKSVW12]. The second and fourth rows of the Ladder column are unpopulated since we were unable to find suitable constituent codes to produce fixed-rate WOM-codes.

The code in the second row of the S.C./Expansion column has symbol-distribution vector $\boldsymbol{p} = (0.521, 0.282, 0.152, 0.0450)$. The code in the third row of the S.C./Expansion column is the result of using Corollary 2 with a two-write fixed-rate code from [SHP12a] with sum-rate 1.5455 and the Spider Code from the second row. The code in the fourth row of the S.C./Expansion column is the result of using the Expansion Construction with the Spider Code from the second row as the constituent. Finally the code in the fifth row of the S.C./Expansion column uses Corollary 2 with a two-write fixed-rate code from [SHP12a] and the code from the fourth row (S.C/Expansion column) as constituents.

2.5 Conclusion

In this chapter, we presented bounds and constructions for WOM-codes that have applications to multi-level Flash memories. Recall from Chapter 1, that the lifetime of a

Flash device is largely a function of the number of times the device is erased. Since WOM-codes reduce the number of times a memory is erased throughout the lifetime of the device, WOM-codes have the potential to extend the lifetime of Flash memory. In addition, our code constructions may have practical relevance since many of them possess short block lengths.

CHAPTER 3

Asymmetric Error-Correction Codes for Flash Memory

3.1 Introduction

In this part of the thesis, we propose codes for Flash memory based upon data collected from a TLC Flash chip. The chapter is organized as follows. In this chapter, the data collected from a TLC Flash chip is summarized and an error mode is proposed. In Section 3.2, code constructions for this model are given. Section 3.3 analyzes the performance of these new codes in terms of their redundancy. In Section 3.4, these constructions are shown through simulation to delay the onset of errors within the memory device longer than traditionally used error correction codes.

3.1.1 Error Characterization of TLC Flash Device

We collected error measurements from sixteen blocks evenly divided across two planes. In a manner analogous to [GCCSYSW], the following testing procedure was repeatedly performed. On the first cycle of every 100 program/erase (P/E) cycles, a block was erased, and random data was then written and finally read back for errors. On the other 99 cycles, the block was simply erased and every page was programmed to simulate the aging of the device.

In Figure 3.1, the Bit Error Rate (BER) is illustrated for the TLC chip tested over the course of its lifetime. It can be seen that over time, the BER increases dramatically.

Furthermore, the increase in error rate varies across the LSB, CSB, and MSB. The 'Symbol Error Rate' curve refers to the symbol error rate when each cell is represented as a symbol over $GF(8)$, and LSB, CSB and MSB curves correspond to the respective bit-wise errors.

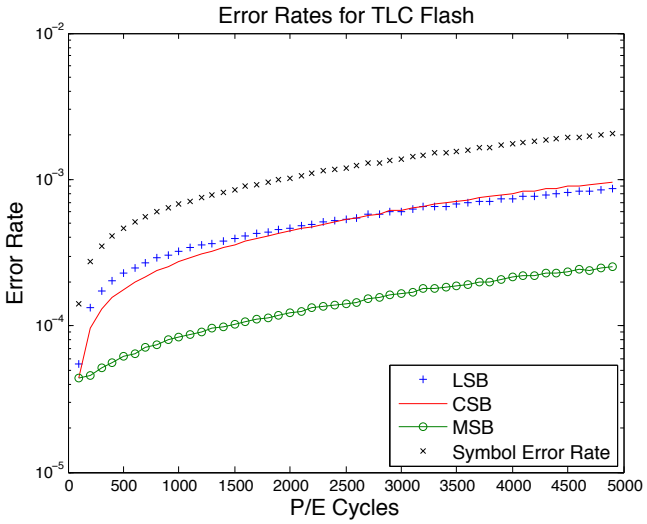

Figure 3.1: Error Rates Measured from a TLC Flash Device.

Figure 3.1 plots is the result of measurements taken from 3.3×10^7 cells monitored across 5000 P/E cycles. The dominant trend from Figure 3.1 is that the 'Symbol Error Rate' appears to be roughly the sum of the individual BERs of the MSB, CSB, and LSB. This suggests that whenever a cell-error occurs, with high probability only one of the three bits in the cell errs. More specifically, as shown in Table 3.1, most of the bit errors in each cell followed a certain pattern. In each row, erroneous bits are marked in red. Table 3.1 also offers an insight on why the BER curves of the LSB and CSB in Figure 3.1 are close to each other: every dominant error of the CSB has an equivalent one of the LSB with almost the same percentage. Upon closer inspection, Table 3.2 shows that 96.17% of cell-errors measured from across the device's lifetime only had a single bit in error.

We note that this distribution of error patterns is a result of the special programming property of the bits where the three bits are not programmed all at once but rather one at a time. This special programming property is due to organization of information in a Flash device: every Flash cell is spread across multiple pages and data is accessed/modified one

page at a time. Suppose the bits are programmed in the following order: first MSB, then CSB, and lastly LSB. Then, a programming error in the MSB causes the LSB and CSB to base their program on the erroneous measurement of the MSB. Thus if the value $(1, 1, 1)$ is written and the MSB is programmed as a 0, then the CSB and LSB will read 0 and will program the cell to its highest level, resulting in the state $(0, 1, 1)$. For more information regarding this property and the mappings of cell values to voltages, see [YSVW11].

The observed distribution is the key motivation for the proposed code constructions that correct a large number of cell-errors with one bit in error and a smaller number of cell errors with more than one bit in error. Note that this error model is considerably different than the one of asymmetric limited magnitude errors, studied in previous works, e.g., [CSBB10] and [KBE11]. That is, the observed errors are not limited in their magnitude as one may expect to see. Interestingly, as shown in the last row of Table 3.1, we observed frequent errors from the lowest level to highest level (cf. Table 1.1). We note that in all these dominant errors in Table 3.1 only one bit was in error.

Table 3.1: Prominent Error Patterns Observed

Programmed state	Errored state	Percentage of errors
000	010	0.2467
000	001	0.2444
111	101	0.0820
111	110	0.0807
000	100	0.0669
011	001	0.0556
100	110	0.0550
011	010	0.0547
100	101	0.0540
111	011	0.0217

Table 3.2: Flash cell error patterns

Number of bits in Flash cell that err	Percentage of errors
1	0.9617
2	0.0314
3	0.0069

3.1.2 Model and Definitions

In this section, the relevant error models as well as code definitions are introduced.

Definition 4. *A linear code $\mathcal{C}$ of length n and dimension k over an alphabet of size q that can correct t or less errors is referred to as an $[n, k, t]_q$ code.*

Here, we assume the ***redundancy*** of a code $\mathcal{C}$ represents the number of parity bits if the code is binary, and, more generally, the number of parity symbols if the code is non-binary. All codes considered in this work have alphabets whose cardinality is $q = 2^m$ where m is some positive integer. Each cell can take on 2^m possible values and is displayed as an m-bit vector. Thus, a word of length n is represented as a length-nm binary vector where bits $mi, \ldots, m(i+1)-1$ represent the ith cell for $1 \leq i \leq n$.

Accordingly, every cell-error is represented as a length-m vector $\boldsymbol{e}_i$. For a fixed ℓ, if $wt(\boldsymbol{e}_i) \leq \ell$ then such an error is called an ***ℓ-bit-cell-error***, where the Hamming weight of a vector $\boldsymbol{x}$ is denoted by $wt(\boldsymbol{x})$. Motivated by the nature of the errors observed, it is useful to define the following class of error-vectors and codes.

Definition 5. *Given the positive integers t and ℓ, an error-vector $\boldsymbol{e} = (\boldsymbol{e}_1, \boldsymbol{e}_1, \ldots, \boldsymbol{e}_n)$ over $(GF(2)^m)^n$ is called a $[t; \ell]_{2^m}$-**bit-error-vector** if the following holds:*

1. *$wt(\boldsymbol{e}) = |\{i : \boldsymbol{e}_i \neq \boldsymbol{0}\}| \leq t$, and*
2. *$\forall i$, $wt(\boldsymbol{e}_i) \leq \ell$.*

Definition 6. *A 2^m-ary linear code $\mathcal{C}$ that can correct every $[t; \ell]_{2^m}$-bit-error-vector is called a $[t; \ell]_{2^m}$-**bit-error-correcting code**.*

From the data collected from the TLC Flash device, it was observed that while most cell-errors suffered a single bit-error, a small number of cells had double or triple bit-errors. Therefore, to correct all observed errors, it is useful to define the following refined error-vectors and corresponding codes.

Definition 7. *Let $0 < \ell_1 < \ell_2 \leq m$, $t_1, t_2 > 0$. Then, a vector $\boldsymbol{e} = (\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_n)$ over $(GF(2)^m)^n$ is called a $[t_1, t_2; \ell_1, \ell_2]_{2^m}$*-**bit-error-vector** *if the following holds:*

1. $wt(\boldsymbol{e}) = |\{i : \boldsymbol{e}_i \neq \boldsymbol{0}\}| \leq t_1 + t_2$,
2. $\forall i$, $wt(\boldsymbol{e}_i) \leq \ell_2$, *and*
3. $|\{i : wt(\boldsymbol{e}_i) > \ell_1\}| \leq t_2$.

The vector (000 001 000 011 000) is an example of a $[1, 1; 1, 2]_{2^3}$-bit-error-vector of length 5, since there are at most $1 + 1 = 2$ cells of non-zero Hamming weight (here these are the second and fourth cell), the maximum weight per cell is 2 (achieved in the fourth cell), and there is at most one cell of weight more than one (again achieved in the fourth cell).

Definition 8. *Let $0 < \ell_1 < \ell_2 \leq m$, $t_1, t_2 > 0$. Then, a 2^m-ary code $\mathcal{C}$ is said to be a $[t_1, t_2; \ell_1, \ell_2]_{2^m}$*-**bit-error-correcting code** *if it can correct every $[t_1, t_2; \ell_1, \ell_2]_{2^m}$-bit-error-vector.*

We collectively refer to codes introduced in Definition 8 as **graded bit-error-correcting codes**. We complete the section with the following definition useful in determining the parity-check matrices of (graded) bit-error-correcting codes.

Definition 9. *Let $A \in GF(q)^{m \times n}$, $B \in GF(q)^{p \times r}$. Then the* ***tensor product*** *of A and B is defined as the matrix*

$$A \otimes B = \begin{pmatrix} a_{1,1}B & \ldots & a_{1,n}B \\ \vdots & \ddots & \vdots \\ a_{m,1}B & \ldots & a_{m,n}B \end{pmatrix} \in GF(q)^{mp \times nr}.$$

Note that the rank of the tensor product can be expressed in terms of the ranks of constituent matrices as,

$$rank(A \otimes B) = rank(A) \cdot rank(B).$$

3.2 Code Constructions

In this section, code constructions are given for (graded) bit-error-correcting codes. The section begins by revisiting a result from [WOL65] that can be used to create $[t;\ell]_{2^m}$-bit-error-correcting codes. This idea is extended to create $[t_1,t_2;\ell_1,\ell_2]_{2^m}$-bit-error-correcting codes.

3.2.1 Tensor Product Codes

We start by presenting a construction of a type of tensor product codes which we also refer to as a $[t;\ell]_{2^m}$-bit-error-correcting code.

Construction A. *(cf. [WOL65]) Let $\mathcal{C}_1$ be an $[m,k_1,\ell]_2$ code with a parity check matrix H_1. Let $\mathcal{C}_2$ be an $[n,k_2,t]_{2^{m-k_1}}$ code with a parity check matrix H_2. Then, the code $\mathcal{C}_A$ with the parity-check matrix*

$$H_A = (H_2 \otimes H_1)\,, \tag{3.1}$$

is a $[t;\ell]_{2^m}$-bit-error-correcting code of length n.

Remark 2. *Note that the parity-check matrix $H_A = (H_2 \otimes H_1)$ corresponds to a binary code of length nm. This matrix can also be converted into a parity-check matrix of a code of length n over $GF(2^m)$ by simply partitioning each row of matrix H_A into consecutive groups of m bits each. Each such group then corresponds to an element in $GF(2^m)$.*

We first provide an example illustrating how these operations are performed.

Example 2. *(cf. [WOL06]) Let H_2 be the following parity check matrix for a $[5,3,1]_4$ code where α is a primitive element from $GF(4)$,*

$$H_2 = \begin{pmatrix} 1 & 0 & 1 & \alpha & \alpha^2 \\ 0 & 1 & 1 & \alpha^2 & \alpha \end{pmatrix}.$$

Let H_1 be a Hamming code of length 3 with the following parity check matrix,

$$H_1 = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}.$$

Representing the elements of $GF(4)$ as $\alpha^0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \alpha^1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \alpha^2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, and $0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ we have

$$H_A = H_2 \otimes H_1 =$$
$$\begin{pmatrix} 1 & \alpha & \alpha^2 & 0 & 0 & 0 & 1 & \alpha & \alpha^2 & \alpha & \alpha^2 & 1 & \alpha^2 & 1 & \alpha \\ 0 & 0 & 0 & 1 & \alpha & \alpha^2 & 1 & \alpha & \alpha^2 & \alpha^2 & 1 & \alpha & \alpha & \alpha^2 & 1 \end{pmatrix}.$$

Using the same symbol-to-binary vector mapping, we can write H_A in the binary representation as

$$H_A = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}.$$

Note that since the parity check matrix of the code $\mathcal{C}_A$ is the tensor product of the matrices H_1 and H_2, and $rank(H_2 \otimes H_1) = rank(H_2) \cdot rank(H_1)$, we get that the redundancy of the code $\mathcal{C}_A$ is $r_1 r_2$, where $r_1 = m - k_1$ and $r_2 = n - k_2$. While the correctness of the error-correction capability was proved in [WOL65], for completeness and in the interest of the subsequent discussion, let us describe here a decoder for the code $\mathcal{C}_A$.

Suppose $\boldsymbol{c} \in \mathcal{C}_A$, where $\boldsymbol{c} = (\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n) \in (GF(2)^m)^n$. Then,

$$
\begin{aligned}
H_A \cdot \boldsymbol{c}^T &= (H_2 \otimes H_1) \cdot \boldsymbol{c}^T \\
&= \begin{pmatrix} h_{1,1}H_1 & \ldots & h_{1,n}H_1 \\ \vdots & \ddots & \vdots \\ h_{r_1,1}H_1 & \ldots & h_{r_1,n}H_1 \end{pmatrix} \cdot \boldsymbol{c}^T \\
&= \begin{pmatrix} h_{1,1} & \ldots & h_{1,n} \\ \vdots & \ddots & \vdots \\ h_{r_1,1} & \ldots & h_{r_1,n} \end{pmatrix} \cdot \begin{pmatrix} H_1 \cdot {\boldsymbol{c}_1}^T \\ \vdots \\ H_1 \cdot {\boldsymbol{c}_n}^T \end{pmatrix},
\end{aligned}
$$

where $h_{i,j}$ represents the symbol in position row i, column j of H_2. Thus, $\boldsymbol{c} \in \mathcal{C}_A$ if and only if $(H_1 \cdot {\boldsymbol{c}_1}^T, \ldots, H_1 \cdot {\boldsymbol{c}_n}^T) \in \mathcal{C}_2$. The code $\mathcal{C}_A$ can be expressed as follows:

$$
\begin{aligned}
\mathcal{C}_A = \{\boldsymbol{c} = (\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n) \in (GF(2)^m)^n \ : \\
(H_1 \cdot \boldsymbol{c}_1^T, \ldots, H_1 \cdot \boldsymbol{c}_n^T) \in \mathcal{C}_2\}.
\end{aligned}
$$

Let

$$\mathcal{D}_1 : \{0,1\}^{r_1} \to \{0,1\}^m, \ \mathcal{D}_2 : (\{0,1\}^{r_1})^{r_2} \to (\{0,1\}^{r_1})^n$$

be the decoders of the codes $\mathcal{C}_1, \mathcal{C}_2$, respectively. Here, and henceforth we assume that the input to each constituent decoder is the syndrome of the received vector and the output is the detected error vector. We note that the input to the constituent decoders will be the corresponding syndrome whereas the input to the decoder of Construction D as well as the upcoming constructions will be the erroneous word. For the ease of the code presentation we use both descriptions of the decoder input. We also assume that if the code can correct t errors, then the weight of the output error vector is at most t. If the decoder finds an error vector of weight greater than t then the all-zero vector is returned as the output.

The decoder $\mathcal{D}_A : (\{0,1\}^m)^n \to (\{0,1\}^m)^n$ of the code $\mathcal{C}_A$ gets as an input a word of

the form $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e}$, where $\boldsymbol{c} \in \mathcal{C}_A$ and $\boldsymbol{e} \in (GF(2)^m)^n$ is a $[t; \ell]_{2^m}$-bit-error-vector. The output of the decoder is the estimate of the error vector, obtained in two steps. First, $\mathcal{D}_2$ produces a set of syndromes $\boldsymbol{s}_i$, $1 \leq i \leq n$, based on the received string $\boldsymbol{y}$. Each of the syndromes $\boldsymbol{s}_i$ serves as the input to $\mathcal{D}_1$ to collectively produce the estimate of the error vector, $\widehat{\boldsymbol{e}}$. The overall decoding algorithm can be described in these two steps:

1. $\mathcal{D}_2(H_2 \cdot (H_1 \cdot \boldsymbol{y}_1^T, \ldots, H_1 \cdot \boldsymbol{y}_n^T)^T) = (\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n)$.

2. $\widehat{\boldsymbol{e}} = (\mathcal{D}_1(\boldsymbol{s}_1), \ldots, \mathcal{D}_1(\boldsymbol{s}_n))$.

Figure 3.2: Decoding map $\mathcal{D}_A$

Lemma 17. *Assume that $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e}$, where $\boldsymbol{c} \in \mathcal{C}_A$ and $\boldsymbol{e} \in (GF(2)^m)^n$ is a $[t; \ell]_{2^m}$-bit-error-vector. The decoder output satisfies $\mathcal{D}_A(\boldsymbol{y}) = \widehat{\boldsymbol{e}} = \boldsymbol{e}$.*

Proof. According to the definition of the code $\mathcal{C}_A$ we have $(H_1 \cdot \boldsymbol{c}_1^T, \ldots, H_1 \cdot \boldsymbol{c}_n^T) \in \mathcal{C}_2$ and we can write

$$\begin{aligned}&(H_1 \cdot \boldsymbol{y}_1^T, \ldots, H_1 \cdot \boldsymbol{y}_n^T)\\ &= (H_1 \cdot \boldsymbol{c}_1^T, \ldots, H_1 \cdot \boldsymbol{c}_n^T) + (H_1 \cdot \boldsymbol{e}_1^T, \ldots, H_1 \cdot \boldsymbol{e}_n^T).\end{aligned}$$

The vector $(H_1 \cdot \boldsymbol{e}_1^T, \ldots, H_1 \cdot \boldsymbol{e}_n^T)$ has weight at most t and since $\mathcal{C}_2$ can correct t errors we get that

$$(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n) = (H_1 \cdot \boldsymbol{e}_1^T, \ldots, H_1 \cdot \boldsymbol{e}_n^T).$$

Next, for every $1 \leq i \leq n$

$$H_1 \cdot \boldsymbol{y}_i^T = H_1 \cdot (\boldsymbol{c}_i^T + \boldsymbol{e}_i^T) = H_1 \cdot \boldsymbol{c}_i^T + H_1 \cdot \boldsymbol{e}_i^T$$

and since $\boldsymbol{s}_i = H_1 \cdot \boldsymbol{e}_i^T$ and the weight of $\boldsymbol{e}_i$ is at most ℓ, we get that $\mathcal{D}_1(\boldsymbol{s}_i) = \boldsymbol{e}_i$, that is, $\boldsymbol{e} = \widehat{\boldsymbol{e}}$. □

3.2.2 Graded Bit-Error-Correcting Codes

The codes given in Construction D correct error patterns according to the maximum number of bit-errors in every cell (or within an m-bit symbol). Recall that in the TLC data it was observed that while most cells suffer a small number of bit-errors, few cell-errors may in fact have a larger number of bit-errors. This observation motivates the following construction of $[t_1, t_2; \ell_1, \ell_2]_{2^m}$-bit-error-correcting codes (also referred to as graded bit-error-correcting codes).

Construction B. *Let $\mathcal{C}_1$ be an $[m, k, \ell_2]_2$ code with a parity check matrix H_1.*

1. *Let $r = m - k$ be such that the following holds:*
   (a) *There exists $0 \leq r' < r$ such that the matrix H_1' comprised of the first r' rows of H_1 is a parity check matrix for an $[m, m - r', \ell_1]_2$ code $\mathcal{C}_1'$,*
   (b) *The matrix H_1'' is an $r'' \times m$ matrix consisting of the last r'' rows of H_1, where $r'' = r - r'$.*
2. *The matrix H_2 is a parity check matrix for an $[n, k_2, t_1 + t_2]_{2^{r'}}$ code $\mathcal{C}_2$, and $r_2 = n - k_2$.*
3. *The matrix H_3 is a parity check matrix for an $[n, k_3, t_2]_{2^{r''}}$ code $\mathcal{C}_3$, and $r_3 = n - k_3$.*

Then, a parity check matrix for a $[t_1, t_2; \ell_1, \ell_2]_{2^m}$-bit-error-correcting code $\mathcal{C}_B$ of length n is

$$H_B = \begin{pmatrix} H_2 \otimes H_1' \\ H_3 \otimes H_1'' \end{pmatrix}. \tag{3.2}$$

Remark 3. *The parity check matrix H_1 of the code $\mathcal{C}_1$ that corrects ℓ_2 errors needs to satisfy the property that it can be decomposed into two matrices H_1' and H_1'', where the first matrix is a parity check matrix of an ℓ_1-error-correcting code $\mathcal{C}_1'$. We note this requirement is not hard to satisfy as many codes can follow this structure, and in particular BCH codes.*

Remark 4. *The resulting parity check matrix in Construction B has the same structure as a parity check matrix for the generalized tensor product code in [IF81]. The focus in [IF81] was to create rate-efficient binary codes with a guaranteed minimum distance and low decoding complexity. Here, the goal is to correct specific error patterns not captured by traditional definitions of minimum distance.*

Before proving the error correction capability of $\mathcal{C}_B$, we provide an illustrative example.

Example 3. *Suppose $\mathcal{C}_1$ is a triple error-correcting $[3,0,3]_2$ code with a parity check matrix $H_1 = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$. Let $r' = 2$ so that $H_1' = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$ is a parity check matrix for a $[3,1,1]_2$ Hamming code and $H_1'' = (\ 1\ 1\ 1\ )$. Let $\mathcal{C}_2$ be a $[15,9,2]_4$ code with a parity check matrix H_2. Furthermore, let $\mathcal{C}_3$ be a $[15,11,1]_2$ Hamming code with a parity check matrix H_3. Then, using Construction B, the code $\mathcal{C}_B$ has a parity check matrix*

$$H_B = \begin{pmatrix} H_2 \otimes H_1' \\ H_3 \otimes H_1'' \end{pmatrix}. \tag{3.3}$$

H_B is a parity check matrix for a $[1,1;1,3]_{2^3}$-bit-error-correcting code. Let α be the primitive element of $GF(4)$. Using the field element representation from Example 1, the matrix H_1' can now be written as $H_1'' = (\ \alpha^0\ \alpha\ \alpha^2\ )$, and the operation $H_2 \otimes H_1'$ can be performed over $GF(4)$. The operation $H_3 \otimes H_1''$ is defined over $GF(2)$. We remark that the particular choice of $\mathcal{C}_1$ in this example results in the same code that was previously proposed in [YSVW11].

Note that $\boldsymbol{c} \in \mathcal{C}_B$ if and only if

$$\mathbf{0} = H_B \cdot \boldsymbol{c}^T = \begin{pmatrix} H_2 \otimes H_1' \\ H_3 \otimes H_1'' \end{pmatrix} \cdot \boldsymbol{c}^T$$
$$= \begin{pmatrix} H_2 \cdot \left(H_1' \cdot {\boldsymbol{c}_1}^T, \ldots, H_1' \cdot {\boldsymbol{c}_n}^T \right)^T \\ H_3 \cdot \left(H_1'' \cdot {\boldsymbol{c}_1}^T, \ldots, H_1'' \cdot {\boldsymbol{c}_n}^T \right)^T \end{pmatrix}.$$

Hence, the code $\mathcal{C}_B$ can be expressed as

$$\mathcal{C}_B = \{\boldsymbol{c} = (\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n) \in (GF(2)^m)^n \ : \\ (H_1' \cdot \boldsymbol{c}_1^T, \ldots, H_1' \cdot \boldsymbol{c}_n^T) \in \mathcal{C}_2, \\ (H_1'' \cdot \boldsymbol{c}_1^T, \ldots, H_1'' \cdot \boldsymbol{c}_n^T) \in \mathcal{C}_3\}.$$

and its redundancy is $r'r_2 + r''r_3$ (see [IF81]).

Let us denote by

$$\mathcal{D}_1 : \{0,1\}^r \to \{0,1\}^m, \ \mathcal{D}_1' : \{0,1\}^{r'} \to \{0,1\}^m,$$
$$\mathcal{D}_2 : (\{0,1\}^{r'})^{r_2} \to (\{0,1\}^{r'})^n,$$
$$\mathcal{D}_3 : (\{0,1\}^{r''})^{r_3} \to (\{0,1\}^{r''})^n,$$

the decoders of the codes $\mathcal{C}_1, \mathcal{C}_1', \mathcal{C}_2$, and $\mathcal{C}_3$, respectively. As before, the input to each decoder is the syndrome and the output is the error vector whose weight is no greater than the guaranteed error-correction capability of the corresponding code.

Before presenting the decoding steps, let us explain the idea behind this construction and its decoding procedure. We start in a similar fashion as the decoder for Construction D, where at most t (now interpreted as $t_1 + t_2$) cell-errors each of weight at most ℓ_1 are found. Clearly, it may not be possible to correct all cell-errors this way since t_2 errors can have weight ℓ_2, and $\ell_2 > \ell_1$. If a cell-error has weight at most ℓ_1 then it is corrected. Otherwise,

it is miscorrected to a cell-error vector with weight at most $\ell_1 + \ell_2$ since the weight of each miscorrection has been restricted to be ℓ_1. This operation in turn guarantees that the new cell-error vector is not a codeword in $\mathcal{C}_1$, since the minimum distance of $\mathcal{C}_1$ is at least $2\ell_2 + 1$. Thus, the next step is to detect the cells which were miscorrected. For cell-errors with more than ℓ_1 bits in error, the remaining part of the syndrome according to the code $\mathcal{C}_1$ is recovered. The decoder $\mathcal{D}_1$ is then used to recover the remaining errors.

The decoder $\mathcal{D}_B : (\{0,1\}^m)^n \to (\{0,1\}^m)^n$ of the code $\mathcal{C}_B$ gets as an input a word of the form $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e}$, where $\boldsymbol{c} \in \mathcal{C}_B$ and $\boldsymbol{e} \in (GF(2)^m)^n$ is a $[t_1, t_2; \ell_1, \ell_2]_{2^m}$-bit-error-vector. The decoder output $\widehat{\boldsymbol{e}}$ is an estimate of the error vector $\boldsymbol{e}$, that is $\mathcal{D}_B(\boldsymbol{y}) = \widehat{\boldsymbol{e}}$. The decoder $\mathcal{D}_B$ performs the following sequence of steps:

1. $\mathcal{D}_2(H_2 \cdot (H_1' \cdot \boldsymbol{y}_1^T, \ldots, H_1' \cdot \boldsymbol{y}_n^T)^T) = (\boldsymbol{s}_1^0, \ldots, \boldsymbol{s}_n^0)$.
2. $\widehat{\boldsymbol{e}}^* = (\mathcal{D}_1'(\boldsymbol{s}_1^0), \ldots, \mathcal{D}_1'(\boldsymbol{s}_n^0))$.
3. $\boldsymbol{y}' = \boldsymbol{y} + \widehat{\boldsymbol{e}}^*$.
4. $\mathcal{D}_2(H_2 \cdot (H_1' \cdot \boldsymbol{y}_1'^T, \ldots, H_1' \cdot \boldsymbol{y}_n'^T)^T) = (\boldsymbol{s}_1', \ldots, \boldsymbol{s}_n')$.
5. $\mathcal{D}_3(H_3 \cdot (H_1'' \cdot \boldsymbol{y}_1'^T, \ldots, H_1'' \cdot \boldsymbol{y}_n'^T)^T) = (\boldsymbol{s}_1'', \ldots, \boldsymbol{s}_n'')$.
6. Let $I = \{i \ : \ (\boldsymbol{s}_i', \boldsymbol{s}_i'') \neq (\boldsymbol{0}, \boldsymbol{0})\}$.
7. Let $\boldsymbol{y}''$ satisfy: $\boldsymbol{y}_i'' = \boldsymbol{y}_i$ if $i \in I$ and $\boldsymbol{y}_i'' = \boldsymbol{y}_i'$ if $i \notin I$.
8. $\mathcal{D}_3(H_3 \cdot (H_1'' \cdot \boldsymbol{y}_1''^T, \ldots, H_1'' \cdot \boldsymbol{y}_n''^T)^T) = (\boldsymbol{s}_1^1, \ldots, \boldsymbol{s}_n^1)$.
9. $\widehat{\boldsymbol{e}} = (\widehat{\boldsymbol{e}}_1, \ldots, \widehat{\boldsymbol{e}}_n)$ where $\widehat{\boldsymbol{e}}_i = \widehat{\boldsymbol{e}}_i^*$ if $i \notin I$ and otherwise $\widehat{\boldsymbol{e}}_i = \mathcal{D}_1(\boldsymbol{s}_i^0, \boldsymbol{s}_i^1)$.

Figure 3.3: Decoding map $\mathcal{D}_B$

Theorem 3. *Assume that $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e}$, where $\boldsymbol{c} \in \mathcal{C}_B$ and $\boldsymbol{e} \in (GF(2)^m)^n$ is a $[t_1, t_2; \ell_1, \ell_2]_{2^m}$-bit-error-vector. The decoder output satisfies $\mathcal{D}_B(\boldsymbol{y}) = \widehat{\boldsymbol{e}} = \boldsymbol{e}$.*

Proof. According to the definition of the code $\mathcal{C}_B$, $(H_1' \cdot \boldsymbol{c}_1^T, \ldots, H_1' \cdot \boldsymbol{c}_n^T) \in \mathcal{C}_2$ and

$$\begin{aligned}&(H_1' \cdot \boldsymbol{y}_1^T, \ldots, H_1' \cdot \boldsymbol{y}_n^T)\\ &= (H_1' \cdot \boldsymbol{c}_1^T, \ldots, H_1' \cdot \boldsymbol{c}_n^T) + (H_1' \cdot \boldsymbol{e}_1^T, \ldots, H_1' \cdot \boldsymbol{e}_n^T).\end{aligned}$$

The vector $(H_1' \cdot \boldsymbol{e}_1^T, \ldots, H_1' \cdot \boldsymbol{e}_n^T)$ has weight at most $t_1 + t_2$. Since the code $\mathcal{C}_2$ can correct $t_1 + t_2$ errors we get that $(\boldsymbol{s}_1^0, \ldots, \boldsymbol{s}_n^0) = (H_1' \cdot \boldsymbol{e}_1^T, \ldots, H_1' \cdot \boldsymbol{e}_n^T)$ after step 1.

At step 2 since for every $1 \leq i \leq n$, $H_1' \cdot \boldsymbol{y}_i^T = H_1' \cdot \boldsymbol{c}_i^T + H_1' \cdot \boldsymbol{e}_i^T$, if $wt(\boldsymbol{e}_i) \leq \ell_1$,

$$\widehat{\boldsymbol{e}}_i^* = \mathcal{D}_1'(\boldsymbol{s}_i^0) = \boldsymbol{e}_i,$$

as $\mathcal{C}_1'$ corrects ℓ_1 errors. However, if the weight of $\boldsymbol{e}_i$ is between $\ell_1 + 1$ and ℓ_2 then $\widehat{\boldsymbol{e}}_i^* = \mathcal{D}_1(\boldsymbol{s}_i^0) \neq \boldsymbol{e}_i$. This observation results from the fact that the decoder for $\mathcal{C}_1'$ can only return a cell-error vector of weight at most ℓ_1. In particular, we get that $wt(\widehat{\boldsymbol{e}}^*) \leq t_2$ and for all $1 \leq i \leq n$, $wt(\widehat{\boldsymbol{e}}_i^*) \leq \ell_1 + \ell_2$. Thus, at the end of step 3, $\boldsymbol{y'}$ contains no cell errors of weight less than ℓ_1 and all the remaining (at most t_2) cell-errors have weight at most $\ell_1 + \ell_2$.

Steps 4 and 5 compute the syndrome using $\boldsymbol{y'}$ as input. Since the minimum distance of the code $\mathcal{C}_1$ is $2\ell_2 + 1 > \ell_1 + \ell_2$, we get that for all $1 \leq i \leq n$, if a miscorrection occurred, then $\widehat{\boldsymbol{e}}_i^*$ is not a codeword in $\mathcal{C}_1$. In such case, $(\boldsymbol{s}_i', \boldsymbol{s}_i'') \neq (\boldsymbol{0}, \boldsymbol{0})$. In step 6, the set I is the set of all i, $1 \leq i \leq n$, such that $\ell_1 < wt(\boldsymbol{e}_i) \leq \ell_2$. In step 7, the word $\boldsymbol{y''}$ is the word $\boldsymbol{y}$ after removing all cell-errors of weight at most ℓ_1.

In step 8 the remaining portion of the syndrome is recovered for all cell-errors with more than ℓ_2 bits in error. Lastly, in step 9 for every cell-error at position i, $1 \leq i \leq n$, if ℓ_1 or fewer bit-errors occurred then $\widehat{\boldsymbol{e}}_i$ is the i-th component of the cell-error vector $\widehat{\boldsymbol{e}}^*$. If more than ℓ_1 bit-errors occurred, the decoder $\mathcal{D}_1$ is used. Since $\mathcal{C}_1$ can correct ℓ_2 errors and the syndrome $H_1 \cdot {\boldsymbol{e}_i}^T = (\boldsymbol{s}_i^0, \boldsymbol{s}_i^1)$ is known for all cell-errors with more than ℓ_1 bits in error, these errors are corrected as well. □

The following example illustrates the decoding process for the preceding construction.

Example 4. *Suppose H_1', H_1'', H_2, H_3, and H_B are as in Example 3. By construction, $\mathcal{C}_B$ is a $[1,1;1,3]_{2^3}$-bit-error-correcting code. Let α be a primitive element of $GF(4)$ with field element representations: $\alpha^0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \alpha^1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \alpha^2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, and $0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$.*

Suppose the all-zero codeword is transmitted and the following vector $\boldsymbol{y}$ is received

$$\boldsymbol{y} = \boldsymbol{e} = (110\ 100\ \mathbf{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}\ ),$$

where $\boldsymbol{e}_1$ has 2 bits in error and $\boldsymbol{e}_2$ has a single bit in error. Here $\mathbf{0}$ is a shorthand for (0 0 0).

In this case, the output of $\mathcal{D}_2$ at step 1 is $(\boldsymbol{s}_1^0, \ldots, \boldsymbol{s}_n^0) = (\ \alpha^2\ \alpha^0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ )$.

The decoder $\mathcal{D}_1'$ at step 2 outputs

$$\widehat{\boldsymbol{e}}^* = (001\ 100\ \mathbf{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}\ ).$$

Notice that for position 2 the output is correct since only a single bit error occurred, whereas for position 1 the decoder $\mathcal{D}_1'$ miscorrects since 2 bits were in error. Now at step 3 the decoder outputs

$$\boldsymbol{y}' = (111\ \mathbf{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}\ ).$$

The output of $\mathcal{D}_2$ at step 4 is clearly the all-zero vector. Now the output of $\mathcal{D}_3$ at step 5 is $(\boldsymbol{s}_1'', \ldots, \boldsymbol{s}_n'') = (1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ )$*, so that an error at position 1 has been detected. Thus, it is known that an error of magnitude greater than 1 has occurred in position 1 (step 6).*

Step 7 results in the following vector

$$\boldsymbol{y}'' = (110\ \mathbf{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}\ ),$$

so that the maximum magnitude of any of the errors is only 2. *The output of* $\mathcal{D}_3$ *at step* 8 *is the all-zero vector of length* 15. *Thus, the last step of the decoding procedure produces*

$$\widehat{\boldsymbol{e}} = (110\ 100\ \mathbf{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}\ ).$$

We now quickly consider a special case where ℓ_2 takes on the maximum possible value.

Special case of $\ell_2 = m$

In the case that $\ell_2 = m$, it is required that $\mathcal{C}_1$ can correct up to m errors. This would imply that a parity check matrix H_1 for such a code is an $m \times m$ matrix with full rank. Thus, given any $[m, m - r', \ell_1]_2$ code $\mathcal{C}_1'$ with a parity matrix H_1', any choice of $m - k$ additional row vectors for H_1'' such that the resulting matrix $H = \begin{pmatrix} H_1' \\ H_1'' \end{pmatrix}$ has full rank can be used to create a $[t_1, t_2; \ell_1, \ell_2]_{2^m}$-bit-error-correcting code $\mathcal{C}_B$. Note that in Construction B, it is required that the parity check matrix H_1 can be decomposed into two parts whereby H_1' is a $[m, m - r', \ell_1]_2$ code. Hence, for $\ell_2 = m$ any $[m, m - r', \ell_1]_2$ code can be chosen for $\mathcal{C}_1'$.

In general, the decoder $\mathcal{D}_1'$ outputs $\mathbf{0}$ if the weight of the error vector output exceeded the error-correction capability of the code. In the current set up with $\ell_2 = m$ such constraint becomes vacuous. As a result, the overall decoder can be simplified as follows.

1. $\mathcal{D}_2(H_2 \cdot (H_1' \cdot \boldsymbol{y}_1^T, \ldots, H_1' \cdot \boldsymbol{y}_n^T)^T) = (\boldsymbol{s}_1^0, \ldots, \boldsymbol{s}_n^0)$.
2. $\boldsymbol{y}' = \boldsymbol{y} + (\mathcal{D}_1'(\boldsymbol{s}_1^0), \ldots, \mathcal{D}_1'(\boldsymbol{s}_n^0))$.
3. $\mathcal{D}_3(H_3 \cdot (H_1'' \cdot \boldsymbol{y}_1'^T, \ldots, H_1'' \cdot \boldsymbol{y}_n'^T)^T) = (\boldsymbol{s}_1^1, \ldots, \boldsymbol{s}_n^1)$.
4. Let $I = \{i \ : \boldsymbol{s}_i^1 \neq 0\}$.
5. $\widehat{\boldsymbol{e}} = (\widehat{\boldsymbol{e}}_1, \ldots, \widehat{\boldsymbol{e}}_n)$ where $\widehat{\boldsymbol{e}}_i = \mathcal{D}_1'(\boldsymbol{s}_i^0)$ if $i \notin I$ and otherwise $\widehat{\boldsymbol{e}}_i = \mathcal{D}_1(\boldsymbol{s}_i^0, \boldsymbol{s}_i^1)$.

Figure 3.4: Decoding map $\mathcal{D}_B$ for $\ell_2 = m$ case

Note that since H_1 has full rank, any of the miscorrections introduced in step 3 are corrected in steps 4 and 5.

3.2.3 Some Extensions

Modification G and Modification 2 presented in this section are extensions of Construction B. The idea is to use a combination of codes whose abilities are to correct errors, correct erasures, and detect errors.

In Modification G, the code $\mathcal{C}_1'$ in Construction B is modified such that it corrects ℓ_1 errors and detects when there are between $\ell_1 + 1$ and ℓ_2 errors. Accordingly, the code $\mathcal{C}_3$ in Construction B needs only to correct t_2 erasures instead of t_2 errors.

Modification 1. *Let $\mathcal{C}_C$ be a code with the following modifications with respect to the code construction of $\mathcal{C}_B$:*

1. *The matrix H_1' now consists of the first r' rows of H_1 where*
   (a) *H_1' is a parity check matrix for an $[m, m - r', \ell_1]_2$-bit-error-correcting code $\mathcal{C}_1'$,*
   (b) *The minimum distance of $\mathcal{C}_1'$ is at least $\ell_1 + \ell_2 + 1$ so the code can also detect an error vector of weight between $\ell_1 + 1$ and ℓ_2.*
2. *H_3 is a parity check matrix of an $[n, k_3, \lceil \frac{t_2}{2} \rceil]_{2^{r''}}$ code $\mathcal{C}_3$ that can correct at least t_2 erasures, and $r_3 = n - k_3$.*

Note that, as before, the matrix H_2 is a parity check matrix for an $[n, k_2, t_1 + t_2]_{2^{r'}}$ code $\mathcal{C}_2$, and $r_2 = n - k_2$. The matrix H_1'' is an $r'' \times m$ matrix consisting of the last r'' rows of H_1, where $r'' = r - r'$.

As before, a parity check matrix for $\mathcal{C}_C$ is $H_C = \begin{pmatrix} H_2 \otimes H_1' \\ H_3 \otimes H_1'' \end{pmatrix}$. The decoders of the codes $\mathcal{C}_1'$ and $\mathcal{C}_3$ are changed while the decoders for $\mathcal{C}_1''$ and $\mathcal{C}_2$ remain the same as in Construction B.

The decoder $\mathcal{D}_1'$, in addition to correcting ℓ_1 errors, also detects if the number of errors is between $\ell_1 + 1$ and ℓ_2. The decoder performs the mapping

$$\mathcal{D}_1' : \{0,1\}^{r'} \to \{0,1\}^m \cup \{E\},$$

where the symbol E indicates a detected error of weight between $\ell_1 + 1$ and ℓ_2. Note that now the decoder $\mathcal{D}_1'$ never miscorrects. The input to the decoder $\mathcal{D}_3$ is no longer a syndrome but a vector $\boldsymbol{x} = (\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{n-1})$ with at most t_2 erasures so that

$$\mathcal{D}_3 : (\{0,1\}^{r''} \cup ?)^n \to (\{0,1\}^{r''})^n,$$

where ? is the erasure symbol. The output of the decoder $\mathcal{D}_C$ is the 'erasure' vector $\boldsymbol{e} = (\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n)$, where for every $\boldsymbol{e}_i \neq \boldsymbol{0}$, $\boldsymbol{e}_i$ is the correct value for $\boldsymbol{x}_i$. The decoder $\mathcal{D}_C : (\{0,1\}^m)^n \to (\{0,1\}^m)^n$ is summarized below. Recall that the input to is $\mathcal{D}_C$ is $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e}$.

1. $\mathcal{D}_2(H_2 \cdot (H_1' \cdot \boldsymbol{y}_1^T, \ldots, H_1' \cdot \boldsymbol{y}_n^T)^T) = (\boldsymbol{s}_1^0, \ldots, \boldsymbol{s}_n^0)$.
2. $\widehat{\boldsymbol{e}}^* = (\mathcal{D}_1'(\boldsymbol{s}_1^0), \ldots, \mathcal{D}_1'(\boldsymbol{s}_n^0))$.
3. Let $I = \{i \ : \ \widehat{\boldsymbol{e}}_i^* = \ E\}$.
4. Let $\widehat{\boldsymbol{e}}'$ satisfy: $\widehat{\boldsymbol{e}}_i' = \boldsymbol{0}$ if $i \in I$ and $\widehat{\boldsymbol{e}}_i' = \widehat{\boldsymbol{e}}_i^*$ if $i \notin I$.
5. $\boldsymbol{y}' = \boldsymbol{y} + \widehat{\boldsymbol{e}}'$.
6. Let $\boldsymbol{x}$ satisfy: $\boldsymbol{x}_i = ?$ if $i \in I$ and $\boldsymbol{x}_i = H_1'' \cdot \boldsymbol{y}_i'^T$ if $i \notin I$.
7. $\mathcal{D}_3(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = (\boldsymbol{s}_1^1, \ldots, \boldsymbol{s}_n^1)$.
8. $\widehat{\boldsymbol{e}} = (\widehat{\boldsymbol{e}}_1, \ldots, \widehat{\boldsymbol{e}}_n)$ where $\widehat{\boldsymbol{e}}_i = \widehat{\boldsymbol{e}}_i^*$ if $i \notin I$ and otherwise $\widehat{\boldsymbol{e}}_i = \mathcal{D}_1(\boldsymbol{s}_i^0, \boldsymbol{s}_i^1 + H_1'' \cdot \boldsymbol{y}_i'^T)$.

Lemma 18 establishes the correctness of the code construction by outlining the decoding procedure as previously done for Construction D and Construction B.

Lemma 18. *The code $\mathcal{C}_C$ is an $[t_1, t_2; \ell_1, \ell_2]_{2^m}$-bit-error-correcting code.*

Proof. Using the same arguments as in the proof of Theorem 3, if $wt(\boldsymbol{e}_i) \leq \ell_1$, then at step 2 we get that

$$\widehat{\boldsymbol{e}}_i^* = \mathcal{D}_1(\boldsymbol{s}_i^0) = \boldsymbol{e}_i.$$

However, if $\ell + 1 \leq wt(\boldsymbol{e}_i) \leq \ell_2$ then since $\mathcal{C}_1'$ can detect up to ℓ_2 errors,

$$\widehat{\boldsymbol{e}}_i^* = \mathcal{D}_1'(\boldsymbol{s}_i^0) = E.$$

Thus, at step 3 we find all the locations of cell-errors with weight greater than ℓ_1. In step 5 all the cell-errors with ℓ_1 or less bits in error are corrected using the output from $\mathcal{D}_1'$ derived in step 2.

In step 6, for every cell-error that has between $\ell_1 + 1$ and ℓ_2 bits in error, $\boldsymbol{x}_i$ =?, and by assumption there are at most t_2 of these. Since $\mathcal{C}_3$ is a code that can correct up to t_2 erasures, then every cell with between $\ell_1 + 1$ and ℓ_2 bits in error is corrected at step 7. Then, as in the proof of Theorem 3 each $\boldsymbol{e}_i$ can be recovered using either $\widehat{\boldsymbol{e}}_i^*$ or the decoder $\mathcal{D}_1$ with the syndrome $(\boldsymbol{s}_i^0, \boldsymbol{s}_i^1 + H_1'' \cdot \boldsymbol{y}_i'^T)$. □

Yet another modification of Construction B is possible that may require less redundancy for the case where $t_2 > t_1$. This modification is described below as Modification 2.

Modification 2. *Let $\mathcal{C}_D$ be a code with these modifications with respect to the code construction of $\mathcal{C}_B$:*

1. *Let the matrix H_1' be a parity check matrix of an ℓ_2-error-detecting code and consisting of the first r' rows of H_1.*

2. *The matrix H_3 is a parity check matrix for a $[n, k_3, \lceil \frac{t_1+t_2}{2} \rceil]_{2^{r''}}$ code $\mathcal{C}_3$ that can correct up to $t_1 + t_2$ erasures, and $r_3 = n - k_3$.*

As before, the matrix H_1'' is an $r'' \times m$ matrix consisting of the last r'' rows of H_1, where $r'' = r - r'$. The matrix H_2 is a parity check matrix for an $[n, k_2, t_1 + t_2]_{2^{r'}}$ code $\mathcal{C}_2$, and $r_2 = n - k_2$.

The parity check matrix is again the concatenation of constituent tensor products, as in (3.3). The encoders and decoders are the same as in Construction B except that the decoder $\mathcal{D}_1'$ can now only detect errors of weight at most ℓ_2. The decoding procedure is outlined below.

1. $\mathcal{D}_2(H_2 \cdot (H_1' \cdot \boldsymbol{y}_1^T, \ldots, H_1' \cdot \boldsymbol{y}_n^T)^T) = (\boldsymbol{s}_1^0, \ldots, \boldsymbol{s}_n^0)$.
2. $\widehat{\boldsymbol{e}}^* = (\mathcal{D}_0'(\boldsymbol{s}_1^0), \ldots, \mathcal{D}_1'(\boldsymbol{s}_n^0))$.
3. Let $I = \{i \ : \ \widehat{\boldsymbol{e}}_i^* = E\}$.
4. Let $\boldsymbol{s}'$ satisfy: $\boldsymbol{s}_i' = ?$ if $i \in I$ and $s_i' = H_1'' \cdot \boldsymbol{y}_i'^T$ if $i \notin I$.
5. $\mathcal{D}_3(\boldsymbol{s}_1', \ldots, \boldsymbol{s}_n') = (\boldsymbol{s}_1^1, \ldots, \boldsymbol{s}_n^1)$.
6. $\widehat{\boldsymbol{e}} = (\mathcal{D}_1(\boldsymbol{s}_1^0, \boldsymbol{s}_1^1 + H_1'' \cdot \boldsymbol{y}_1'^T), \ldots, \mathcal{D}_1(\boldsymbol{s}_n^0, \boldsymbol{s}_n^1 + H_1'' \cdot \boldsymbol{y}_n'^T))$.

Lemma 19. *The code $\mathcal{C}_D$ is a $[t_1, t_2; \ell_1, \ell_2]_{2^m}$-bit-error-correcting code.*

Proof. Suppose $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e}$ is the input to decoder $\mathcal{D}_D$ where $\boldsymbol{c} \in \mathcal{C}_D$ and $\boldsymbol{e} \in (GF(2)^m)^n$ is a $[t_1, t_2; \ell_1, \ell_2]_{2^m}$-bit-error-vector. Then, since $\mathcal{C}_2$ can correct $t_1 + t_2$ errors,

$$(\boldsymbol{s}_1^0, \ldots, \boldsymbol{s}_n^0) = (H_1' \cdot \boldsymbol{e}_1^T, \ldots, H_1' \cdot \boldsymbol{e}_n^T),$$

where $\boldsymbol{e} = (\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n)$. If $0 < wt(\boldsymbol{e}_i) \leq \ell_2$,

$$\widehat{\boldsymbol{e}}_i^* = \mathcal{D}_1'(\boldsymbol{s}_i^1) = \ E$$

since $\mathcal{D}_1'$ can detect up to ℓ_2 errors. Thus, at step 3 the locations of all the cell-errors is noted. In step 5 the locations of these errors is passed to $\mathcal{D}_3$ and since $\mathcal{D}_3$ can correct $t_1 + t_2$

erasures, the remaining portion of the syndrome $\boldsymbol{s}_i^1$ is recovered. At step 6 these two pieces are combined and given to $\mathcal{D}_1$ which can correct up to ℓ_2 errors so that every cell error can now be corrected. □

3.3 Code Analysis

In this section, we analyze the performance of Constructions D and B as well as Modifications G and 2 with respect to their redundancy. We begin by considering $[t, \ell]_{2^m}$ -bit-error-correcting codes and show that under certain conditions Construction D is perfect. We then analyze the minimum required redundancy of $[t_1, t_2; \ell_1, \ell_2]_{2^m}$-bit-error-correcting codes by counting the number of error vectors. Note that every $[t_1 + t_2; \ell_2]_{2^m}$-bit-error-correcting code, given by Construction D, is a $[t_1, t_2; \ell_1, \ell_2]_{2^m}$-bit-error-correcting code as well. Hence, we will analyze under what conditions for the parameters $n, m, t_1, t_2, \ell_1, \ell_2$ Constructions D or B and Modifications G or 2 provide the least redundancy.

A nice property of Construction D is that if the codes $\mathcal{C}_1$ and $\mathcal{C}_2$ from Construction D are perfect then the code $\mathcal{C}_A$ is perfect as well. For completeness, we provide the proof of this statement since we could not find it elsewhere.

Lemma 20. *If the codes $\mathcal{C}_1$ and $\mathcal{C}_2$ from Construction D are perfect, then the code $\mathcal{C}_A$ is perfect as well.*

Proof. The number of $[t; \ell]_{2^m}$-bit-error-vector of length n is given by

$$\sum_{k=0}^{t} \binom{n}{k} \left(\sum_{i=1}^{\ell} \binom{m}{i} \right)^k.$$

Since the code $\mathcal{C}_1$ is a perfect binary ℓ-error-correcting code we get

$$2^{k_1} \cdot \sum_{i=0}^{\ell} \binom{m}{i} = 2^m, \tag{3.4}$$

and similarly, since $\mathcal{C}_2$ is a perfect t-error-correcting code over $GF(2^{m-k_1})$ we get

$$(2^{m-k_1})^{k_2} \cdot \sum_{k=0}^{t} \binom{n}{k} (2^{m-k_1}-1)^k = (2^{m-k_1})^n. \tag{3.5}$$

From (3.4) we get $2^{m-k_1} - 1 = \sum_{i=1}^{\ell} \binom{m}{i}$ and thus (3.5) becomes

$$(2^{m-k_1})^{k_2} \cdot \sum_{k=0}^{t} \binom{n}{k} \left(\sum_{i=1}^{\ell} \binom{m}{i} \right)^k = (2^{m-k_1})^n,$$

or

$$\sum_{k=0}^{t} \binom{n}{k} \left(\sum_{i=1}^{\ell} \binom{m}{i} \right)^k = 2^{(m-k_1)(n-k_2)} = 2^{r_1 r_2}.$$

Therefore, the code $\mathcal{C}_A$ is perfect as well. □

The number of different $[t_1, t_2; \ell_1, \ell_2]_{2^m}$-bit-error-vectors is given in the next lemma.

Lemma 21. *Let m be a positive integer and $0 < \ell_1 < \ell_2 \leq m$, $t_1, t_2 > 0$. Then the number of different $[t_1, t_2; \ell_1, \ell_2]_{2^m}$-bit-error-vectors of length $n \geq t_1 + t_2$ is given by*

$$V = \sum_{i=0}^{t_2} \binom{n}{i} \left(\sum_{k=\ell_1+1}^{\ell_2} \binom{m}{k} \right)^i \cdot \sum_{j=0}^{t_1+t_2-i} \binom{n-i}{j} \left(\sum_{p=1}^{\ell_1} \binom{m}{p} \right)^j.$$

Proof. Let i be the number of cells that have between $\ell_1 + 1$ and ℓ_2 bits in error. The total number of such errors is t_2. Then for any value of $i, 0 \leq i \leq t_2$ there are $\binom{n}{i}$ choices of positions for such errors to occur. There are $\sum_{k=\ell_1+1}^{\ell_2} \binom{m}{k}$ distinct cell-errors that have more than ℓ_1 bits in error but at most ℓ_2 bits in error, for each such symbol error.

Similarly, let j be the number of cells that have between 1 and ℓ_1 bits in error. Then for any value of $j \leq t_1 + t_2 - i$ there are $\binom{n-i}{j}$ possible choices of cells where errors can occur. Furthermore, when an error occurs there are $\sum_{p=1}^{\ell_1} \binom{m}{p}$ distinct cell-errors that have between 1 and ℓ_1 bits in error. □

Since V denotes the volume of the error vectors in a $[t_1, t_2; \ell_1, \ell_2]_{2^m}$ bit-error-correcting codes, $r_{min} = \lceil \log V \rceil$ is the minimal redundancy in bits. Here and in the rest of the chapter all logarithms are in base 2. In general, the redundancy of Constructions A and B and Modifications 1 and 2 depends on the choice of constituent codes. However, to simplify the analysis we assume in this section that both n and m are relatively large so that $\mathcal{H}(\frac{t}{n})$ can be reasonably approximated as $t \log n$, where $\mathcal{H}$ denotes the binary entropy function. Thus, all the analysis in this section will be performed using such approximations.

The goal of the following is to identify regimes where one code construction does better than the other. From Lemma 21 we approximate r_{min} as

$$r_{min} \approx (t_1 + t_2) \log n + (t_1\ell_1 + t_2\ell_2) \log m.$$

The redundancy of the code $\mathcal{C}_B$ in Construction B using optimal choices redundancy-wise for matrices H_1, H_2, and H_3 is

$$\begin{aligned} r_B \approx & (t_1 + t_2)(\log n + r') + t_2(\log n + r'') \\ \approx & (t_1 + 2t_2) \log n + (t_1 + t_2)\ell_1 \log m + t_2(\ell_2 - \ell_1) \log m \\ = & (t_1 + 2t_2) \log n + (t_1\ell_1 + t_2\ell_2) \log m, \end{aligned}$$

where $r' = \ell_1 \log m$ and $r'' = r - r' \approx t_2(\ell_2 - \ell_1) \log m$ from Construction B.

Using the approximations for the redundancy of Construction B given above, we now consider the difference between r_{min} and the redundancy of the code specified by Construction B. This difference is approximately

$$r_B - r_{min} \approx t_2 \log n.$$

Another alternative to constructing a $[t_1, t_2; \ell_1, \ell_2]_{2^m}$-bit-error-correcting code is by using a $[t_1 + t_2; \ell_2]_{2^m}$-bit-error-correcting code $\mathcal{C}_A$ from Construction D. The redundancy of $\mathcal{C}_A$ is

approximately

$$r_A \approx (t_1 + t_2)(\log n + \ell_2 \log m).$$

Therefore, the code $\mathcal{C}_B$ outperforms the code $\mathcal{C}_A$ redundancy-wise approximately when $r_B \lesssim r_A$, that is,

$$(t_1 + 2t_2) \log n + (t_1\ell_1 + t_2\ell_2) \log m \lesssim (t_1 + t_2)(\log n + \ell_2 \log m),$$

or

$$\frac{\log n}{\log m} \lesssim \frac{t_1}{t_2}(\ell_2 - \ell_1).$$

Thus, Construction B requires less redundancy than Construction D whenever the ratio $\frac{t_1}{t_2}(\ell_2 - \ell_1)$ is large.

Now, we consider the redundancy of a code given by Modification G. Since a t_2 erasure-correcting code of length-n over $2^{r''}$ can correct at least $\lfloor \frac{t_2}{2} \rfloor$ errors, the redundancy can be approximated as $\frac{t_2}{2}(\log n + r'')$ bits. Then the redundancy of $\mathcal{C}_C$ is

$$r_C \approx (t_1 + t_2)(\log n + r') + \frac{t_2}{2}(\log n + r'').$$

Since a binary code that can correct up to ℓ_1 errors and detect up to ℓ_2 errors can correct at least $\ell_1 + \lfloor \frac{\ell_2 - \ell_1}{2} \rfloor$ errors, r' and r'' are approximated as $\frac{\ell_1 + \ell_2}{2} \log m$ and $\frac{\ell_2 - \ell_1}{2} \log m$, respectively. Then we have

$$r_C \approx (t_1 + \frac{3}{2}t_2) \log n + \frac{1}{4}\left(2t_1(\ell_1 + \ell_2) + t_2(\ell_1 + 3\ell_2)\right) \log m.$$

There are now two alternatives to consider. If Construction D is used to create a $[t_1 + t_2; \ell_2]_{2^m}$ code $\mathcal{C}_A$, then $r_A \gtrsim r_C$ approximately when

$$\frac{\log n}{\log m} \lesssim (\ell_2 - \ell_1)\left(\frac{t_1}{t_2} + \frac{1}{2}\right).$$

If Construction B is used to create a $[t_1, t_2; \ell_1, \ell_2]_{2^m}$ code $\mathcal{C}_B$, then $r_B \gtrsim r_C$ approximately when

$$\frac{\log n}{\log m} \gtrsim (\ell_2 - \ell_1)\left(\frac{t_1}{t_2} - \frac{1}{2}\right).$$

Thus, Modification G offers the least redundancy amongst Constructions D and B and Modification G whenever the two previous inequalities hold. It will be shown in Example 5 that this modification also does well in the special case where $t_1 = t_2$ and $\ell_2 = \ell_1 + 1$ for any n, m. In general, the code with the least redundancy from amongst the choices $\mathcal{C}_A, \mathcal{C}_B$, and $\mathcal{C}_C$ depends on the ratio $\frac{\log n}{\log m}$. Table 3.3 shows under what conditions for $\frac{\log n}{\log m}$, $\mathcal{C}_A$, $\mathcal{C}_B$, or $\mathcal{C}_C$ requires the least redundancy.

Table 3.3: Comparison of conditions for least redundancies for $\mathcal{C}_A, \mathcal{C}_B$, and $\mathcal{C}_C$

Code	Condition on $\frac{\log n}{\log m}$
$\mathcal{C}_A$	$> (\frac{t_1}{t_2} + \frac{1}{2})(\ell_2 - \ell_1)$
$\mathcal{C}_B$	$< (\frac{t_1}{t_2} - \frac{1}{2})(\ell_2 - \ell_1)$
$\mathcal{C}_C$	$< (\frac{t_1}{t_2} + \frac{1}{2})(\ell_2 - \ell_1)$ and $> (\frac{t_1}{t_2} - \frac{1}{2})(\ell_2 - \ell_1)$

$\mathcal{C}_D$ is omitted from the table above because, as we will see, the optimal choice among $\mathcal{C}_A, \mathcal{C}_B, \mathcal{C}_C$, and $\mathcal{C}_D$ no longer depends on the same constants. We now consider the redundancy of $\mathcal{C}_D$ discussed in Modification 2. A $(t_1 + t_2)$ error-correcting code of length-n over $2^{r'}$ requires approximately $(t_1 + t_2)\log n + (t_1 + t_2)r'$ bits. The redundancy of a $(t_1 + t_2)$ erasure-correcting code over $2^{r''}$ can be approximated as requiring $\frac{t_1+t_2}{2}\log n + \frac{t_1+t_2}{2}r''$ bits. Furthermore, a binary code that can detect up to ℓ_2 errors is approximated as requiring $\frac{\ell_2}{2}\log m$ bits of redundancy. Note that now both r' and r'' are approximately $\frac{\ell_2}{2}\log m$. Then

the redundancy of $\mathcal{C}_D$ is approximately

$$\begin{aligned} r_D &\approx (t_1+t_2)(\log n + r') + \frac{t_1+t_2}{2}(\log n + r'') \\ &= \frac{3(t_1+t_2)}{2}\log n + \frac{3(t_1+t_2)}{4}\ell_2 \log m. \end{aligned}$$

Then, r_D is approximately less than r_A when

$$\frac{\log n}{\log m} \lesssim \frac{\ell_2}{2}.$$

The redundancy of Construction B is $(t_1+2t_2)\log n + (t_1\ell_1 + t_2\ell_2)\log m$ so that $\mathcal{C}_D$ requires approximately less redundancy than $\mathcal{C}_B$ when

$$\frac{\log n}{\log m} \gtrsim \frac{t_1(3\ell_2 - 4\ell_1) - t_2\ell_2}{2(t_2 - t_1)}.$$

The redundancy of $\mathcal{C}_C$ is $(t_1+\frac{3}{2}t_2)\log n + \frac{1}{4}\left(2t_1(\ell_1+\ell_2) + t_2(\ell_1+3\ell_2)\right)\log m$. Thus, using these approximations, $\mathcal{C}_D$ requires less redundancy than $\mathcal{C}_C$ whenever

$$\frac{\log n}{\log m} \lesssim \frac{t_1(2\ell_1 - \ell_2) + t_2\ell_1}{2t_1}.$$

The following example illustrates a special case where Modification G is nearly optimal.

Example 5. *Let $\ell_1 = 1, \ell_2 = 2$ and $t_1 = t_2 = 1$. Suppose $\tilde{\mathcal{C}}$ is a $[15, 7, 2]_2$ code with a parity check matrix $\tilde{H}$ and odd minimum distance. Now we append an extra parity bit to every codeword in $\tilde{\mathcal{C}}$ to get the code $\mathcal{C}_1$. The minimum distance of the code $\tilde{\mathcal{C}}$ is increased by 1 and the resulting $\mathcal{C}_1$ code is a $[16, 7, 2]_2$ code. Suppose H_1 is a parity check matrix of $\mathcal{C}_1$. Let H_1' be the top $r' = 5$ rows of H_1, and let H_1'' be the remaining (bottom) $r'' = 4$ rows of H. Then, H_1' is a parity check matrix for a $[16, 11, 1]_2$ code. Note that this is an extended single error-correcting Hamming code with minimum distance 4 and $r' = 5$.*

Let $\mathcal{C}_2$ be a double error-correcting code over $GF(2)^5$ of length n with parity check matrix H_2, and let $\mathcal{C}_3$ be a single erasure-error-correcting code over $GF(2)^4$ of length n with a parity

check matrix H_3. Using the approach in Modification 1, and the matrices H_1', H_1'', H_2 and H_3 as the constituents in this construction, we construct a $[1,1;1,2]_{2^{16}}$-bit error-correcting code.

The redundancy can be analyzed as follows. Since $\mathcal{C}_2$ is a double error-correcting code over $GF(2)^5$, it requires approximately $2\left(\log(n)+5\right)$ bits of redundancy. Since a single erasure-correcting code $\mathcal{C}_3$ can be created using only a single additional parity symbol, it follows that the redundancy of $\mathcal{C}_3$ is approximately 4 bits. Thus, the overall redundancy of the code in the example is $2\log(n)+14$ bits. Note that from Lemmas 4 and 5, the optimal redundancy is approximately $(t_1+t_2)\log n + t_1\ell_1\log m + t_2\ell_2\log m = 2\log n + 3\log 16 = 2\log n + 12$. The constructed code is thus nearly optimal.

In summary, Construction B offers the least redundancy amongst the four code choices whenever $\frac{t_1}{t_2}$ is large and $\ell_2 - \ell_1$ is large. Modification 2 has the least redundancy whenever $t_2 >> t_1$. However, if $t_1 >> t_2$ and $(\ell_2-\ell_1)\left(\frac{t_1}{t_2}-\frac{1}{2}\right) \leq \frac{\log n}{\log m} \leq (\ell_2-\ell_1)\left(\frac{t_1}{t_2}+\frac{1}{2}\right)$, then Modification G has approximately less redundancy than the other three code options. Construction D requires approximately the least redundancy for large $\frac{\log n}{\log m}$.

3.4 Performance and Results

In this section, the performance of various linear error-correcting codes with guaranteed error-correction capability is evaluated for a TLC Flash device. The goal of the simulations was to evaluate the ability of different error-correcting codes to delay the onset of errors for as long as possible. We compared five different types of codes:

1. binary codes with the same error correction capability for the LSB, CSB, and MSB pages,

2. binary codes with different error correction capability for the LSB, CSB, and MSB pages,

3. non-binary codes over $GF(4)$ applied to the CSB and LSB sharing the same physical cells and binary codes applied to the MSB,

4. non-binary codes over $GF(8)$ which correct errors in a group of LSB, CSB, and MSB pages sharing the same physical cells,

5. graded bit-error-correcting codes.

These codes were constructed for lengths of 4096, 8192, and 16384 bits, and for rates between approximately 0.86 and 0.89. All the codes we used in our constructions are based on binary and non-binary BCH codes. The codes listed below were created using the same techniques as in the MinT [SS14] database.

The specification of the parameters for all different types of codes are summarized in the next tables.

1. Binary codes with the same error correction capability for the LSB, CSB, and MSB pages.

Code	Rate
$[2^{12}, 3531, 47]_2$	0.86
$[2^{13}, 7242, 73]_2$	0.88
$[2^{14}, 14591, 128]_2$	0.89

2. Binary codes (labeled 'Binary Codes' in Figures) with different error correction capability for the LSB, CSB, and MSB pages.

LSB Code	CSB Code	MSB Code	rate
$[2^{12}, 3351, 62]_2$	$[2^{12}, 3339, 63]_2$	$[2^{12}, 3915, 15]_2$	0.86
$[2^{13}, 6904, 99]_2$	$[2^{13}, 6891, 100]_2$	$[2^{13}, 7931, 20]_2$	0.88
$[2^{14}, 13905, 177]_2$	$[2^{14}, 13863, 180]_2$	$[2^{14}, 15963, 30]_2$	0.89

3. Scheme A - A non-binary code over $GF(4)$ applied to the CSB and LSB pages sharing the same physical cells and a binary code applied to the MSB page.

CSB, LSB Code	MSB Code	Rate
$[2^{12}, 3338, 84]_4$	$[2^{12}, 3915, 15]_2$	0.86
$[2^{13}, 6882, 125]_4$	$[2^{13}, 7931, 20]_2$	0.89
$[2^{14}, 13862, 240]_4$	$[2^{14}, 15963, 30]_2$	0.89

4. Non-binary codes over $GF(8)$ which correct errors in a group of LSB, CSB, and MSB pages sharing the same physical cells.

Code	Rate
$[2^{12}, 3534, 80]_8$	0.86
$[2^{13}, 7244, 120]_8$	0.88
$[2^{14}, 14593, 205]_8$	0.89

5. Graded bit-error-correcting codes applied to the LSB, CSB, and MSB pages. The following table lists the constituent codes that comprise each code depicted in the figures that follow. More specifically, the $[81, 7; 1, 3]_{2^3}$ code (shown in Figures 3.5 and 3.6) is the result of applying Construction B to the two constituents in the first row of the table. The $[120, 8; 1, 3]_{2^3}$ (shown in Figures 3.7 and 3.8) code is the result of applying Construction B to the two constituents from the second row of the table. The $[242, 8; 1, 3]_{2^3}$ (shown in Figures 3.9 and 3.10) code is the result of applying Construction B to the two constituents in the third row.

Codes	Rate
$[2^{12}, 3302, 88]_4, [2^{12}, 4011, 7]_2$	0.86
$[2^{13}, 6847, 128]_4, [2^{13}, 8087, 8]_2$	0.89
$[2^{14}, 13757, 250]_4, [2^{14}, 16271, 8]_2$	0.89

We assume that all the constructed codes in our simulations have efficient encoders in terms of their time complexity as they are based on BCH codes. Hence the complexity of the resulting graded bit-error-correcting code is on the same order as the complexity of a BCH code.

For each of the codes we used, if its error correction capability is t errors and there are at most t errors, then they are all corrected. Otherwise, we assume that the decoder would detect the error and will leave the information word unchanged.

For the codes that read multiple pages at the same time, it is assumed that if an error occurs in the decoder then a page error occurs across all the pages from which the symbol was read. This means that when a symbol error occurs under a 2-bit alphabet (i.e., where the symbol was read using the information from two pages), both pages are considered to err. Similarly, when a symbol error occurs under a 3-bit alphabet it is assumed that three pages are in error. Likewise, whenever a symbol error occurs, it is assumed all the bits comprising the symbol are in error.

In Figures 3.5 and 3.6 we show the page error rate and the bit error rate, respectively, for codes of length 4096. Figures 3.7 and 3.8 display the page error rate and the bit error rate for codes of length 8192. In Figures 3.9 and 3.10 the page error rate and the bit error rate is plotted for codes of length 16384. We note that in each performance comparison plots, the code lengths are the same.

As can be seen in Figures 3.5 through 3.10 below, the codes over $GF(8)$ and the basic binary codes (where the same binary code is applied to the LSB, CSB, MSB) consistently have the highest error rates both at a bit level and a page level. The $GF(8)$ code is inefficient since it can correct any type of cell error despite the fact that when a cell error occurs there is usually only one bit in error. The basic binary code is clearly inefficient since it does not take advantage of the difference in error rate between the MSB, CSB, and LSB. The remaining codes have lower error rates because they capitalize on these error profile characteristics.

Among the codes tested, the graded-bit-error correcting codes delayed the onset of errors the longest. Scheme A performed slightly worse than the method of applying different binary codes to each bit line. Part of the reason for this trend is that the binary codes used were simply better codes in the sense that they are closer to the sphere-packing bound than the non-binary codes tested in this setup. Despite this advantage, the graded bit-error codes still outperformed all the binary codes tested.

Remark 5. *We quickly remark on the potential performance gain offered by optimal codes (i.e., codes that meet the sphere-packing bound). To see the potential performance gain by using optimal codes (codes that they are close to the sphere-packing bound), we ran simulations with the following codes:*

1. $[89, 8; 1, 3]_{2^3}$ *of length* 4096*,*
2. $[150, 8; 1, 3]_{2^3}$ *of length* 8192*, and*
3. $[280, 12; 1, 3]_{2^3}$ *of length* 16384*.*

Using the same approximations as in the analysis section, these codes have rates of around $0.88, 0.90$*, and* 0.90*, respectively. Using the first code, no errors appeared in the tested device until cycle* 4700*. The second code delayed the onset of errors until cycle* 4400 *and the third code delayed errors until cycle* 4600*. Thus, the device lifetime can be further extended if even better constituent codes were to be discovered and used.*

While the simulations support the benefits of the proposed approach, it should also be noted that the proposed coding scheme will require a change in the current architecture of Flash memories. Currently, in order to reduce the interferences among pages, it is required to write and read each one of the MSB, CSB, and LSB pages independently. However, independent coding of pages which share the same physical cells does not take advantage of the correlation between errors. Our goal in this chapter is to demonstrate this correlation among errors and to propose suitable coding schemes that take the advantage of these

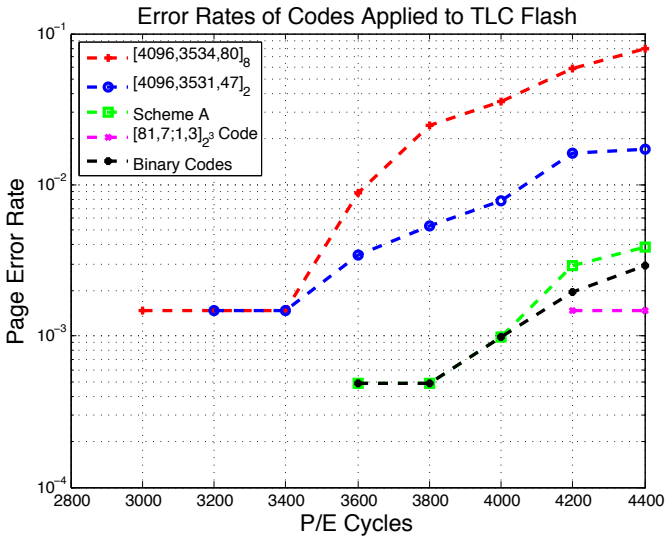

Figure 3.5: Page Error Rates of Codes Applied to TLC Flash for Code Lengths 512B.

correlations. The coding gain is demonstrated in the analysis of the codes in Section 3.3 and by simulation in this section. Our coding scheme only requires the three pages to be written together. This can be easily accomplished in case big files are written or by extending the logical page size. We are also aware that this change might decrease the reading and writing speed of the memory and thus we suggest to use this coding scheme only towards the end of the lifetime of the memory.

We quickly note that once errors were observed, the BER of the our graded bit-error-correcting codes scheme was in some cases inferior to the BER of the 'Binary Codes' scheme. This property results from the joint coding of three pages in our scheme. In particular, once the codes fail to correct the errors, more bits and pages were affected (and therefore more errors occurred). Furthermore, the binary codes are close to the sphere packing bound while the non-binary codes we used in our scheme were far from optimal. In fact, Remark 5 shows that if optimal non-binary codes were to be used as constituents in the graded-bit-error correcting code design, then even better results could be achieved.

In this section, it was demonstrated that leveraging graded bit-error-correcting codes delays the onset of high error rates associated with multilevel Flash devices. Because graded

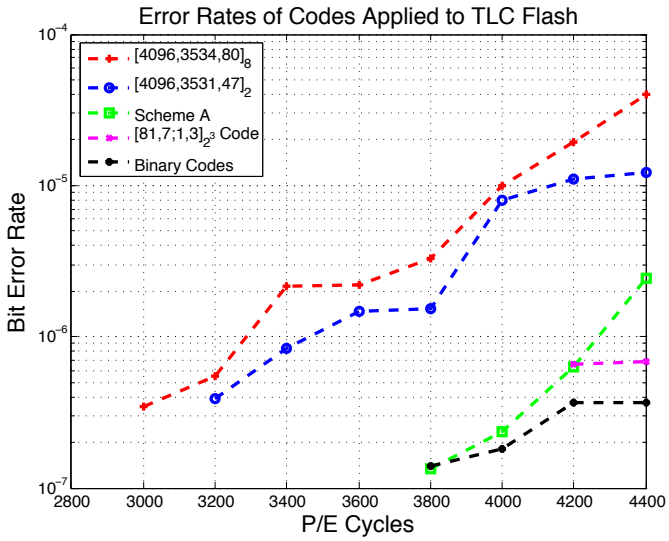

Figure 3.6: Bit Error Rates of Codes Applied to TLC Flash for Code Lengths 512B.

bit-error-correcting codes are designed specifically to account for the asymmetric nature of the observed errors, these codes can prolong the lifetime of multilevel Flash memory devices.

3.5 Conclusion

Recall, from Chapter 1 that as a result of the programming process, when an error occurs in a TLC Flash cell typically only a single bit of information (of the three total bits) is in err. This observation was used to motivate new error-correcting code constructions based upon generalized tensor product codes. The proposed codes were analytically and empirically shown to offer a potentially valuable component for future coding schemes in the context of Flash memory. In particular, the codes could potentially extend the lifetime of Flash memory by delaying the onset of errors.

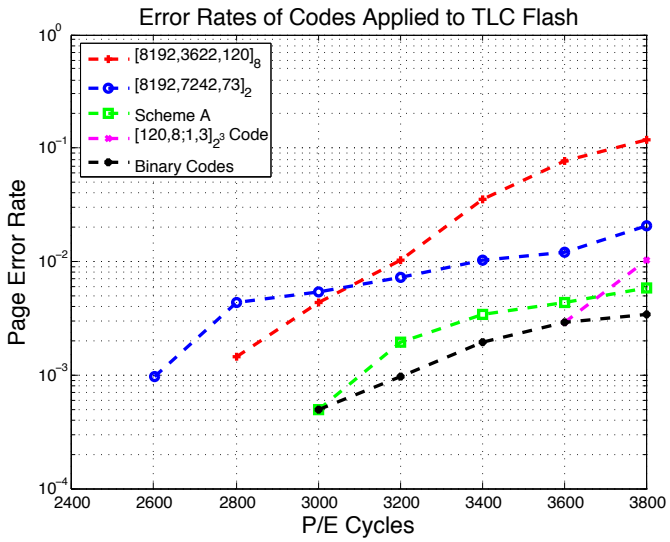

Figure 3.7: Page Error Rates of Codes Applied to TLC Flash for Code Lengths 1KB.

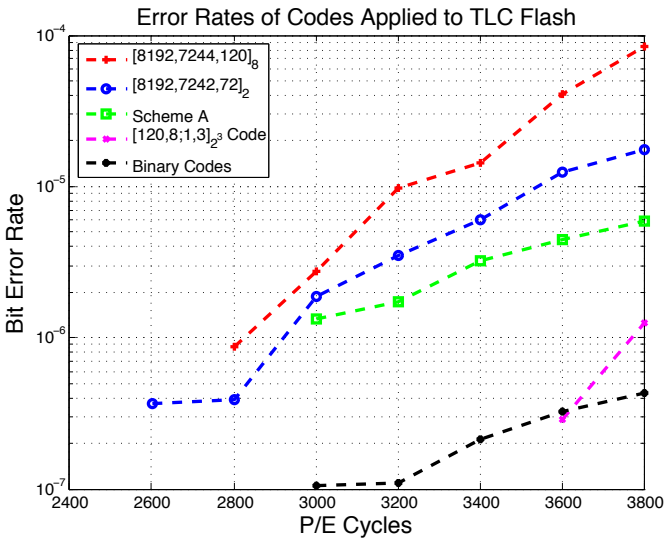

Figure 3.8: Bit Error Rates of Codes Applied to TLC Flash for Code Lengths 1KB.

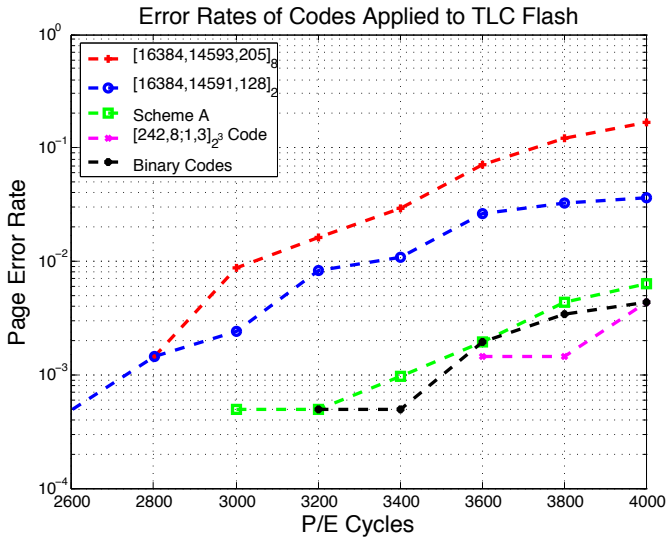

Figure 3.9: Page Error Rates of Codes Applied to TLC Flash for Code Lengths 2KB.

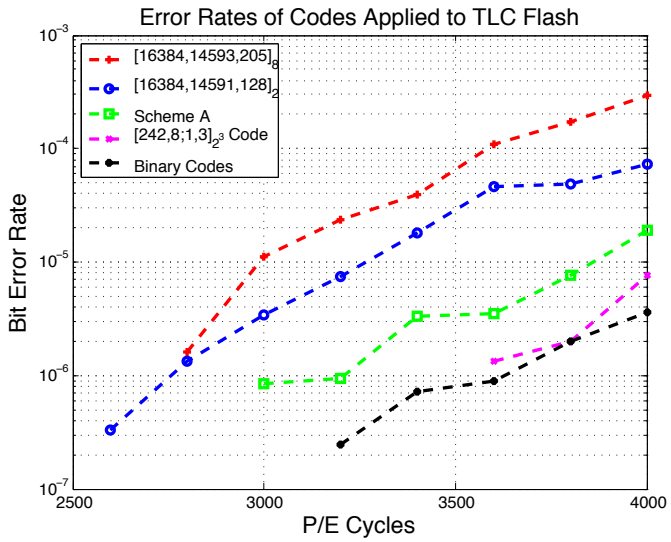

Figure 3.10: Bit Error Rates of Codes Applied to TLC Flash for Code Lengths 2KB.

CHAPTER 4

Synchronization Codes for Rank Modulation Systems

In this chapter, we consider coding for synchronization errors in a rank modulation system. The chapter is organized as follows. In this section, we formally define the erasures and deletions models studied in this chapter and review the Kendall's τ, Ulam, and Hamming distance metrics over permutations. In Section 4.1, we show how to use codes in these three distance metrics in order to construct codes for our erasures and deletions models. In Section 4.2, we give a construction of codes in the Ulam distance which improves upon the best known ones. In Section 4.5, we study codes capable of correcting unstable deletions.

Let $\mathbb{S}_n$ denote the set of all $n!$ permutations of n elements, chosen to be $\{1, 2, \ldots, n\}$. We use the vector notation to denote a permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$. Given some permutation $\pi = (\pi_1, \pi_2, \ldots, \pi_n) \in \mathbb{S}_n$, its inverse permutation is $\pi^{-1} = (\pi_1^{-1}, \pi_2^{-1}, \ldots, \pi_n^{-1})$, where π_i^{-1} is the location of the element i in π. For example, for $\pi = (6, 1, 3, 2, 5, 4)$ we have $\pi^{-1} = (2, 4, 3, 6, 5, 1)$. The set $\{1, \ldots, n\}$ is denoted by $[n]$, and for two positive integers $a < b$, the set $\{a, \ldots, b\}$ is denoted by $[a, b]$.

We first formally define the four models of stable/unstable erasures and deletions. For a permutation $\pi = (\pi_1, \ldots, \pi_n) \in \mathbb{S}_n$, and a set of positions $I \subseteq [n]$, $\pi(I)$ is the set $\pi(I) = \{\pi_i \ : \ i \in I\}$. For an integer $a \in [n]$ and a subset $I \subseteq [n]$, the integer $a(I) \in [n]$ is defined as $a(I) = a - |\{i \in I \ : \ i < a\}|$. For example, assume $\pi = (6, 1, 3, 2, 5, 4)$ and $I = \{1, 4, 5\}$, then $\pi(I) = \{6, 2, 5\}$ and $\pi_3(\pi(I)) = 3(\{6, 2, 5\}) = 3 - 1 = 2$.

Definition 10. *Assume that $\pi = (\pi_1, \ldots, \pi_n)$ is a permutation in $\mathbb{S}_n$ and $I \subseteq \{1, \ldots, n\}$ is*

a positions set of size t. We consider the following four models of erasures and deletions:

1. ***Stable Erasure (SE)**: The permutation π suffered t stable erasures (SEs) in the positions set I, resulting in the vector $\pi' = (\pi'_1, \ldots, \pi'_n)$, if*

   (a) for $i \in I$, $\pi'_i = ?$, and

   (b) for $i \in [n] \setminus I$, $\pi'_i = \pi_i$.

2. ***Unstable Erasure (UE)**: The permutation π suffered t unstable erasures (UEs) in the positions set I, resulting in the vector $\pi' = (\pi'_1, \ldots, \pi'_n)$, if*

   (a) for $i \in I$, $\pi'_i = ?$, and

   (b) for $i \in [n] \setminus I$, $\pi'_i = \pi_i(\pi(I))$.

3. ***Stable Deletion (SD)**: The permutation π suffered t stable deletions (SDs) in the positions set I, resulting in the vector $\pi' = (\pi'_1, \ldots, \pi'_{n-t})$, if*
   for $k \in [n] \setminus I$ and $i = k(I)$, $\pi'_i = \pi_k$.

4. ***Unstable Deletion (UD)**: The permutation π suffered t unstable deletions (UDs) in the positions set I, resulting in the permutation $\pi' = (\pi'_1, \ldots, \pi'_{n-t}) \in \mathbb{S}_{n-t}$, if*
   for $k \in [n] \setminus I$ and $i = k(I)$, $\pi'_i = \pi_k(\pi(I))$.

*A code $\mathcal{C} \subseteq \mathbb{S}_n$ is called a **t-SE/UE/SD/UD-correcting code** if it can correct at most t SEs/UEs/SDs/UDs, respectively.*

The next example illustrates these four models.

Example 6. *Let $\pi = (6, 1, 3, 2, 5, 4) \in \mathbb{S}_6$ and $I = \{1, 4, 5\}$. Then, the following vectors are the received ones for each model:*

1. *Stable Erasure: $\pi' = (?, 1, 3, ?, ?, 4)$.*

2. *Unstable Erasure: $\pi' = (?, 1, 2, ?, ?, 3)$.*

3. *Stable Deletion:* $\pi' = (1, 3, 4)$.

4. *Unstable Deletion:* $\pi' = (1, 2, 3)$.

Note that the first model in Definition 10 is the easiest one and the last one is the hardest one, with respect to the amount of information that is lost. The last model of an unstable deletion, has an analogous model known as an ***unstable insertion*** such that more symbols can be inserted and the other cells scale their value accordingly. For example, assume that the symbol 2 is inserted between the first and second cells in the permutation π from Example 6. Then, the read permutation in $\mathbb{S}_7$ will be $(7, 2, 1, 4, 3, 6, 5)$.

The reading errors in Definition 10 assume that a certain symbol was not read properly and thus was either deleted or erased. The next studied model assumes that all symbols are read and received, however some symbols are read together so the order between them, and only them, is not known. We define this erasure model formally.

Definition 11. *We say that a permutation $\pi \in \mathbb{S}_n$ suffers an $\boldsymbol{e} = (e_1, \ldots, e_t)$-**soft-erasure** if there are t sets $I_1, \ldots, I_t$, such that for $1 \leq \ell \leq t$, $I_\ell = [a_\ell, b_\ell]$, $b_\ell = a_\ell + e_\ell - 1$, $1 \leq a_1 < b_1 < a_2 < b_2 < \cdots < a_t < b_t \leq n$, and only the orders of the elements in each of the t groups $\pi(I_1), \ldots, \pi(I_t)$ are not known. The **erasure-weight** of the soft-erasure $\boldsymbol{e}$ is $\omega(\boldsymbol{e}) = \sum_{\ell=1}^{t} \binom{e_\ell}{2}$. A code $\mathcal{C} \subseteq \mathbb{S}_n$ is called an E-**soft-erasure-correcting code** if it can correct any $\boldsymbol{e}$-soft-erasure of erasure-weight at most E.*

Our approach in finding codes for the aforementioned models is to use, when possible, some of the already existing results of error-correcting codes over permutations. There are several different metrics that these codes were studied for and the following metrics are the ones we use in this work.

An adjacent transposition in a permutation $\pi \in \mathbb{S}_n$ is the local exchange of two adjacent elements in π. The ***Kendall's*** τ distance [KG90] between two permutations $\sigma, \pi \in \mathbb{S}_n$ is denoted by $d_\tau(\sigma, \pi)$ and is defined to be the minimum number of adjacent transpositions required to obtain the permutation π from the permutation σ. The ***Hamming distance***

between two permutations $\pi, \sigma \in \mathbb{S}_n$, denoted by $d_H(\pi, \sigma)$, is defined as the number of positions for which π and σ differ. For two permutations $\pi, \sigma \in \mathbb{S}_n$, let $\ell(\pi, \sigma)$ be the length of a longest common subsequence of π and σ. The ***Ulam distance*** between π and σ is defined as $d_\circ(\pi, \sigma) = n - \ell(\pi, \sigma)$ [BSU72, DH98, FSM13].

Example 7. *Let* $\pi = (4, 3, 1, 2, 5)$ *and* $\sigma = (4, 3, 5, 1, 2)$*, then* $d_\tau(\pi, \sigma) = 2$*,* $d_H(\pi, \sigma) = 3$*, and* $d_\circ(\pi, \sigma) = 1$*.*

The minimum distance of a code, according to any metric, is the minimum distance between every two codewords in the code. For a code $\mathcal{C} \subseteq \mathbb{S}_n$, its minimum Kendall's τ, Hamming, Ulam distance is denoted by $d_\tau(\mathcal{C})$, $d_H(\mathcal{C})$, $d_\circ(\mathcal{C})$, respectively.

4.1 Codes from Existing Codes in other Metrics

In this section we present codes for the erasures and deletions models defined in the previous section. In particular, we show how codes in the Kendall's τ, Hamming, and Ulam distances can be used for these models.

Let us start with stable erasures. First notice that if only a single stable erasure occurred then it is immediate to complete the missing symbol since its location is known by the erasure as well as its value, which is the missing symbol in the permutation. Thus, the code $\mathbb{S}_n$ is a single-SE-correcting code. For more than a single stable erasure, we show in the next theorem how codes in the Hamming distance are necessary and sufficient in the SE model.

Theorem 4. *A code* $\mathcal{C} \subseteq \mathbb{S}_n$ *is a* t*-SE-correcting code if and only if* $d_H(\mathcal{C}) \geq t + 1$*.*

Proof. We show that if $d_H(\mathcal{C}) \geq t + 1$ then $\mathcal{C}$ is a t-SE-correcting code. Assume in the contrary that $\mathcal{C}$ is not a t-SE-correcting code. Thus, there are two permutations $\pi, \sigma \in \mathcal{C}$ and two positions sets $I_1, I_2 \subseteq [n]$, each of size at most t, such that if π' is the result of stable erasures in the positions set I_1 in π and σ' is the result of stable erasures in the positions set I_2 in σ, then $\pi' = \sigma'$. First notice that $I_1 = I_2$. Otherwise, assume without loss of generality that there exists $i \in I_1 \setminus I_2$, so we get $\pi'_i =?$ and $\sigma'_i \neq ?$ which is a contradiction. Hence, we

can denote $I_1 = I_2 = I$ and $|I| \leq t$. Then we get that for every $i \in [n] \setminus I$, $\pi_i = \pi'_i = \sigma'_i = \sigma_i$, and thus $d_H(\pi, \sigma) \leq |I| \leq t$, in contradiction.

The second part holds since we can consider $\mathcal{C}$ as a code in $[n]^n$ so its minimum Hamming distance has to be at least $t + 1$. □

We next move to the models of unstable erasures and stable deletions. Note that in unstable erasures the locations but not the values are not known, while in stable deletions the values but not locations are not known. The next lemma establishes a property claiming that these two models are equivalent.

Lemma 22. *A permutation $\pi \in \mathbb{S}_n$ suffered t UEs if and only if its inverse permutation π^{-1} suffered t SDs.*

Proof. Let $\pi = (\pi_1, \ldots, \pi_n)$ and $\pi^{-1} = (\pi_1^{-1}, \ldots, \pi_n^{-1})$. Assume that π suffered t UEs in the positions set I, resulting in the vector $\pi' = (\pi'_1, \ldots, \pi'_n)$. Let $\pi'^{-1} = (\pi'^{-1}_1, \ldots, \pi'^{-1}_{n-t})$ be a length-$(n-t)$ vector which specifies the locations of the $(n-t)$ non-erased symbols in π'. For every $k \in [n] \setminus I$, $\pi'_k = \pi_k(\pi(I))$ and hence the location of $i \in [n-t]$ in π' is the location of the symbol k_i in π such that $i = k_i(\pi(I))$. That is, for $i \in [n-t]$, $\pi'^{-1}_i = \pi^{-1}_{k_i}$, where $i = k_i(\pi(I))$. Therefore, π'^{-1} is the vector received from π^{-1} after having t stable deletions in the positions set $\pi(I)$.

To prove the other direction, since $(\pi^{-1})^{-1} = \pi$, we can simply prove that if π suffered t SDs then π^{-1} suffered t UEs. If so, assume that π suffered t SDs in the positions set I, resulting with the vector $\pi' = (\pi'_1, \ldots, \pi'_{n-t})$, where for $i \in [n-t]$, $\pi'_i = \pi_{k_i}$, where $i = k_i(I)$. Let $\pi'^{-1} = (\pi'^{-1}_1, \ldots, \pi'^{-1}_n)$ be a length-n vector which specifies the locations of the symbols $[n]$ in π' or specifies a ? in case that the symbol does not appear in π'. That is, for $i \in \pi(I)$ have $\pi'^{-1}_i =$? and for $i \notin \pi(I)$, $\pi'^{-1}_i = \pi_i^{-1}(I)$, or, since $\pi^{-1}(\pi(I)) = I$, $\pi'^{-1}_i = \pi_i^{-1}(\pi^{-1}(\pi(I)))$. Hence, the vector π'^{-1} equals the vector received from π^{-1} after having t UEs in the positions set $\pi(I)$. □

As a result of Lemma 22 we conclude the following corollary.

Corollary 7. *There exists a t-UE-correcting code of cardinality M if and only if there exists a t-SD-correcting code of cardinality M.*

To complete this discussion we only need to consider codes in one of these two models. We will show how codes in the Ulam distance are necessary and sufficient for codes in the SD model.

Theorem 5. *A code $\mathcal{C} \subseteq \mathbb{S}_n$ is a t-SD-correcting code if and only if $d_\circ(\mathcal{C}) \geq t+1$.*

Proof. We show that if $\mathcal{C}$ is a t-SD-correcting code then $d_\circ(\mathcal{C}) \geq t+1$. Assume in the contrary that $d_\circ(\mathcal{C}) \leq t$ and let $\pi, \sigma \in \mathcal{C}$ be such that $d_\circ(\pi, \sigma) = t' \leq t$. Hence, π and σ have a common subsequence of length $\ell(\pi, \sigma) = n - d_\circ(\pi, \sigma) = n - t' \geq n - t$ and let S be the symbols in this common subsequence. Let I_π be the positions set of the symbols $[n] \setminus S$ in π and similarly let I_σ be the positions set of the symbols $[n] \setminus S$ in σ. Then, if π' is the result of t SDs in π in the positions set I_π and σ' is the result of t SDs in σ in the positions set I_σ, we get that $\pi' = \sigma'$. Therefore, the code $\mathcal{C}$ is not a t-SD-correcting code, which is a contradiction.

In order to prove the other direction, we show that if $d_\circ(\mathcal{C}) \geq t+1$ then $\mathcal{C}$ is a t-SD-correcting code. Assume in the contrary that $\mathcal{C}$ is not a t-SD-correcting code. Thus, there are two permutations $\pi, \sigma \in \mathcal{C}$ and two positions sets $I_1, I_2 \subseteq [n]$, each of size at most t, such that if π' is the result of stable deletions in the positions set I_1 in π and σ' is the result of stable deletions in the positions set I_2 in σ, then $\pi' = \sigma'$. First we have that $|I_1| = |I_2|$ and since $\pi' = \sigma'$, π and σ have a common subsequence of length $n - |I_1| \geq n - t$. Therefore, $d_\circ(\pi, \sigma) \leq n - (n - t) = t$, in contradiction again. □

Lastly in this section we turn to handle the soft-erasure model. The connection between this model of erasures and the Kendall's τ-metric is established in the next theorem.

Theorem 6. *Let $\mathcal{C} \subseteq \mathbb{S}_n$ be a code where $d_\tau(\mathcal{C}) \geq E+1$. Then, $\mathcal{C}$ is an E-soft-erasure-correcting code.*

Proof. Assume to the contrary that $\mathcal{C}$ is not an E-soft-erasure-correcting code. Then, there exist two permutations π, σ, and two soft-erasures $\boldsymbol{e}^{\pi} = (e_1^{\pi}, \ldots, e_t^{\pi})$, $\boldsymbol{e}^{\sigma} = (e_1^{\sigma}, \ldots, e_t^{\sigma})$ of erasure-weights at most E such that the vectors received after each of the two soft-erasures are the same. Let $I_1^{\pi}, \ldots, I_t^{\pi}$ be the sets of erasures in π and similarly, $I_1^{\sigma}, \ldots, I_t^{\sigma}$ are the sets of erasures in σ. First note that $I_j^{\pi} = I_j^{\sigma}$ for $1 \leq j \leq t$, since otherwise the received vectors, as a result of the soft-erasures, are not the same. From the same reason, the symbols in each of the t groups are the same for π and σ and the order of all other symbols in π and σ is the same. Therefore, there can be only $\sum_{\ell=1}^{t} \binom{e_{\ell}^{\pi}}{2}$ pairs of symbols that π and σ do not agree on. However, that means that the Kendall's τ distance between π and σ is at most $\sum_{\ell=1}^{t} \binom{e_{\ell}^{\pi}}{2} \leq E$, which is a contradiction. □

4.2 New Codes for the Ulam Metric

In this section, we present new codes for the Ulam metric. In the first subsection, we introduce some notation, tools, and codes that will be used in the following subsection to describe the code construction for the Ulam metric. We proceed by describing a code over $\mathbb{S}_n$ that can correct a prescribed number of stable deletions. As a consequence of Theorem 5, these codes are also suitable for the Ulam metric. Lastly, we will comment on how our construction improves upon the state of the art codes for this metric from [FSM13]. For the remainder of this section, we use the terms deletion(s) and stable deletion(s) interchangeably. Furthermore, we use the terms erasure(s) and stable erasure(s) interchangeably as well.

4.2.1 Notation and Auxiliary Codes

For a sequence π with elements from the set $[n] \cup \{?\}$ and for $s \in [n]$, let $D(\pi, s)$ be the result of removing all occurrences of the symbol s in π. If s is not contained in π, then $D(\pi, s) = \pi$. More generally, for $\mathcal{I} \subset [n]$, $D(\pi, \mathcal{I})$ is the result of removing all symbols from $\mathcal{I}$ in π. Notice that in this case, the set $\mathcal{I}$ contains the symbols to be deleted and not the locations of those symbols. For $\mathcal{I}' \subset [n]$, let $\sigma = Er(\pi, \mathcal{I}')$ be the result of substituting the

elements of $\mathcal{I}'$ in π with the symbol ?.

The following code can determine the locations of deletions given that the locations of the deletions satisfy certain constraints, that will be explained later. For the remainder of this section, we assume that n, ℓ are positive integers where $n > \ell$ and $\ell | n$. We define

$$\mathcal{C}_{\ell,n}^{L} = \{\pi \in \mathbb{S}_n : i \in [n],\ \pi_i \equiv i - 1 (\mathrm{mod} \ell)\}.$$

For $\sigma = (\sigma_1, \ldots, \sigma_n) \in \mathbb{S}_n$, the vector $\sigma(\mathrm{mod}\ell) = (\sigma'_1, \ldots, \sigma'_n)$ is defined by $\sigma'_i = \sigma_i(\mathrm{mod}\ell)$ for $1 \leq i \leq n$. Thus, for any codeword $\pi \in \mathcal{C}_{\ell,n}^{L}$, the vector $\pi(\mathrm{mod}\ell)$ is a periodic sequence of length n and period ℓ where each period is $(0, \ldots, \ell - 1)$.

We now describe a decoding algorithm $\mathcal{D}_{\ell,n}^{L}$ for the code $\mathcal{C}_{\ell,n}^{L}$. We note that although the map $\mathcal{D}_{\ell,n}^{L}$ is not formally a decoder, with a slight abuse of notation it will be referred to as such and the procedure below will be referred to as a decoding algorithm. Suppose that $\pi \in \mathcal{C}_{\ell,n}^{L}$ is the stored codeword and σ is the retrieved word, where $\sigma = D(\pi, \mathcal{I})$ with $\mathcal{I} \subset [n]$, $|\mathcal{I}| = t < n$. The input to $\mathcal{D}_{\ell,n}^{L}$ is σ.

1. Initialize $j = 0$ and let $\zeta^{(1)} = (\sigma, 0)$.
2. Let $j = j + 1$.
3. Let i_j be the smallest i, where $1 \leq i \leq n - t + j$, such that $\zeta_i^{(j)} \not\equiv i - 1 (\mathrm{mod}\, \ell)$ and $\zeta_i^{(j)} \neq ?$. If no such symbol exists, go to step 5).
4. Let $\zeta^{(j+1)}$ be the result of inserting the symbol ? into $\zeta^{(j)}$ at position i_j. Go to step 2).
5. Define $\hat{\pi} = (\zeta_1^{(j)}, \ldots, \zeta_{n-t+j-1}^{(j)})$.

Figure 4.1: Decoding map $\mathcal{D}_{\ell,n}^{L}$

We illustrate the decoder $\mathcal{D}_{n,\ell}^{L}$ with the following example.

Example 8. *Suppose* $n = 12$ *and* $\ell = 3$ *and let* $\pi = (3, 7, 5, 9, 1, 8, 6, 4, 2, 12, 10, 11) \in$

$\mathcal{C}^L_{3,12}$ *and let* $\mathcal{I} = \{1, 3, 8, 11\}$. *Then,* $\sigma = (7, 5, 9, 6, 4, 2, 12, 10)$. *In this case,* $\zeta^{(1)} = (7, 5, 9, 6, 4, 2, 12, 10, 0)$, $\zeta^{(2)} = (?, 7, 5, 9, 6, 4, 2, 12, 10, 0)$, $\zeta^{(3)} = (?, 7, 5, 9, ?, 6, 4, 2, 12, 10, 0)$, $\zeta^{(4)} = (?, 7, 5, 9, ?, ?, 6, 4, 2, 12, 10, 0)$, *and* $\zeta^{(5)} = (?, 7, 5, 9, ?, ?, 6, 4, 2, 12, 10, ?, 0)$, *and* $\hat{\pi} = (?, 7, 5, 9, ?, ?, 6, 4, 2, 12, 10, ?)$.

Let $b(\pi, \mathcal{I}, \ell)$ be equal to 1 if there exists a subset of ℓ symbols from $\mathcal{I}$ that appear consecutively in π. Otherwise, $b(\pi, \mathcal{I}, \ell) = 0$. If $b(\pi, \mathcal{I}, \ell) = 1$, then let $\ell(\pi, \mathcal{I})$ be the set of symbols that constitute the first occurrence of a subset of ℓ symbols from $\mathcal{I}$ that appear consecutively in π. If $b(\pi, \mathcal{I}, \ell) = 0$, then let $\ell(\pi, \mathcal{I}) = \emptyset$. For example, let π be as in Example 8, then, $b(\pi, \{3, 7, 5, 8, 6, 4\}, 3) = 1$ and $3(\pi, \{3, 7, 5, 8, 6, 4\}) = \{3, 5, 7\}$.

The following claim follows from the decoding algorithm for $\mathcal{C}^L_{\ell,n}$. For a sequence $\boldsymbol{x}$ with symbols from $[n] \cup \{?\}$ let $|\boldsymbol{x}|$ denote the length of $\boldsymbol{x}$.

Claim 10. *For positive integers* n, ℓ, *suppose* $\pi \in \mathcal{C}^L_{\ell,n}$, $\mathcal{I} \subset [n]$, *and* $\sigma = D(\pi, \mathcal{I})$. *If* $b(\pi, \mathcal{I}, \ell) = 0$, *then* $\mathcal{D}^L_{\ell,n}(\sigma) = Er(\pi, \mathcal{I})$. *Otherwise,* $|\mathcal{D}^L_{\ell,n}(\sigma)| < |\pi| = n$.

In words, if the longest maximal substring deleted from π has length less than ℓ, that is $b(\pi, \mathcal{I}, \ell) = 0$, then the output of the decoder is π with symbols of $\mathcal{I}$ replaced by ?, which provides the locations of the deleted symbols. Otherwise, the decoder returns a sequence with length shorter than the length of π.

For our code construction in the next subsection we will use two more codes which are described as follows. The first code is a code capable of correcting a single stable deletion over permutations. Let $\mathcal{C}^E_{n'} \subseteq \mathbb{S}_{n'}$ be a code that can correct a single stable deletion from [LEV91] and let $\mathcal{D}^E_{n'}$ be a decoder for this code[LEV91]. The decoder $\mathcal{D}^E_{n'}$ operates as follows. Suppose $\sigma = D(\pi, s)$, where $\pi \in \mathcal{C}^E_{n'}$. The output of $\mathcal{D}^E_{n'}(\sigma)$ is the ordered triplet (s, s', s'') where s is the symbol deleted from π to obtain σ, s' is the symbol immediately before s in π, and s'' is the symbol immediately after s in π. If s is the final symbol in π, then $s'' = 0$ and similarly if s is the first symbol in π, then $s' = 0$.

The second code will be a code in the Hamming metric. Let $\mathcal{C}^H_{d,n} \subseteq \mathbb{S}_n$ be a code with minimum Hamming distance d. From Theorem 4, there exists a decoder $\mathcal{D}^H_{d,n} : ([n] \cup \{?\})^n \rightarrow$

$\mathbb{S}_n$ for $\mathcal{C}^H_{d,n}$ that can correct up to $d-1$ stable erasures, where the location of the erasures are represented by the symbol ?.

4.2.2 Code Construction

Using the tools from the previous subsection, we now present a code capable of correcting $2\ell-1$ stable deletions. The idea will be to combine the constraints for the codes $\mathcal{C}^L_{\ell,n}$, $\mathcal{C}^E_{n/\ell}$, and $\mathcal{C}^H_{3\ell-2,n}$. For a permutation $\pi \in \mathbb{S}_n$ and an integer $0 \leq i \leq \ell-1$, let $\pi_{\pi \equiv_\ell i}$ be the subsequence of π that only contains the symbols from the set $\{s \in [n] : s \equiv i (\mathrm{mod} \ell)\}$. For integers m, n, k, and $\boldsymbol{x} \in [n]^m$, let $(\boldsymbol{x}-k)/\ell = (y_1, \ldots, y_m)$ be such that for $1 \leq i \leq m$, $y_i = \lfloor (x_i - k)/\ell \rfloor$.

Construction C. *For positive integers n, ℓ where $n > \ell$ and $\ell | n$, let $\mathcal{C}^U_{2\ell,n} \subseteq \mathbb{S}_n$ be the code consisting of all permutations $\pi \in \mathbb{S}_n$ that satisfy the following conditions:*

1. $\pi \in \mathcal{C}^H_{3\ell-2,n}$
2. $\pi \in \mathcal{C}^L_{\ell,n}$, *and*
3. $(\pi_{\pi \equiv_\ell i} - i)/\ell \in \mathcal{C}^E_{n/\ell}$ *for* $0 \leq i \leq \ell-1$.

We will show that the code $\mathcal{C}^U_{2\ell,n}$ has Ulam distance at least 2ℓ by showing that it can recover from any $m = 2\ell-1$ deletions. Suppose $\sigma = D(\pi, \mathcal{I}) \in [n]^{n-m}$ where $\mathcal{I} \subset [n]$, $|\mathcal{I}| = m$, and $\pi \in \mathcal{C}^U_{2\ell,n}$.

We first outline the decoding procedure. We will first attempt to determine the locations of the deletions using the decoder $\mathcal{D}^L_{\ell,n}$. From Claim 10, $\mathcal{D}^L_{\ell,n}$ can determine the locations of all the deletions except if $b(\pi, \mathcal{I}, \ell) = 1$. Recall that if $b(\pi, \mathcal{I}, \ell) = 1$, then there was a substring deleted from π whose length is at least ℓ. In this case, the decoder $\mathcal{C}^E_{n/\ell}$ is used to determine the location where the substring (of length at least ℓ) was deleted from π and the locations of the remaining deletions are discovered with the aid of $\mathcal{D}^L_{\ell,n}$. Finally, using $\mathcal{D}^H_{3\ell-2,n}$ the values of the deleted symbols are recovered.

We now turn to formally define the decoding procedure. We refer to the decoding map for $\mathcal{C}^U_{2\ell,n}$ as $\mathcal{D}^U_{2\ell,n} : [n]^{n-m} \to \mathbb{S}_n$. The input to the map is σ and the output is an estimate $\hat{\pi}$ of the codeword $\pi \in \mathcal{C}^U_{2\ell,n}$. The decoder is presented in Fig. 4.2, where we use the following notation. Let $s', s'' \in [n]$. For a vector $\sigma \in [n]^{n-m}$, we refer to the set of symbols in σ that are between the symbols s', s'', exclusive, as $\sigma[s', s'']$. If $s' = 0$, then $\sigma[s', s'']$ is the set of symbols that appear before the symbol s'' in σ. Similarly, if $s'' = 0$, then $\sigma[s', s'']$ is the set of symbols that appear after the symbol s' in σ.

1. Let $\sigma^{(1)} = \mathcal{D}^A_{\ell,n}(\sigma)$. If $|\sigma^{(1)}| = n$, then let $\sigma^{(4)} = \sigma^{(1)}$ and go to step 7). Otherwise, go to step 2).
2. Let k be the smallest non-negative integer such that $|\sigma_{\sigma \equiv_\ell k}| = \frac{n}{\ell} - 1$. Let $\zeta = \sigma_{\sigma \equiv_\ell k}$.
3. Define $(s, s', s'') = \mathcal{D}^E_{n/\ell}(\frac{\zeta - k}{\ell})$ and let $\mathcal{S} = \sigma_{[\ell \cdot s' + k, \ell \cdot s'' + k]}$.
4. Let $\sigma^{(2)}$ be the result of inserting the symbol $\ell \cdot s + k$ immediately before the symbol $\ell \cdot s'' + k$ in σ. Otherwise, let $\sigma^{(2)}$ be the result of appending the symbol $\ell \cdot s + k$ to σ.
5. Define $\sigma^{(3)} = D(\sigma^{(2)}, \mathcal{S})$.
6. Define $\sigma^{(4)} = \mathcal{D}^A_{\ell,n}(\sigma^{(3)})$.
7. Let $\hat{\pi} = \mathcal{D}^H_{3\ell-2,n}(\sigma^{(4)})$.

Figure 4.2: Decoding map $\mathcal{D}^U_{2\ell,n}$ for $\mathcal{C}^U_{2\ell,n}$

Theorem 7. *The code $\mathcal{C}^U_{2\ell,n}$ has Ulam distance at least 2ℓ.*

Proof. The result is proven by showing that the output $\hat{\pi}$ of $\mathcal{D}^U_{2\ell,n}$ equals π. In this proof, let $A = \ell(\pi, \mathcal{I})$. Suppose that at step 1, we have $|\sigma^{(1)}| = n$. Then, from Claim 10, we have $\sigma^{(1)} = Er(\pi, \mathcal{I})$ and $A = \emptyset$. In step 7, since π belongs to a code with minimum Hamming distance $3\ell - 2$, and $|\mathcal{I}| = 2\ell - 1 \leq 3\ell - 2$, the values of the deleted symbols can be recovered. Thus, when $|\sigma^{(1)}| = n$, we have that $\hat{\pi} = \pi$.

We now suppose that $|\sigma^{(1)}| = n - \ell$, which is the only other possibility for if $A \neq \emptyset$, then $|A| = \ell$. Since $2\ell - 1$ elements are deleted from π, by the pigeon hole principle, there exists k such that $|\sigma_{\sigma \equiv_\ell k}| = \frac{n}{\ell} - 1$ (the algorithm arbitrarily picks the smallest possible value for k in step 2). The deleted symbol from $\pi_{\pi \equiv_\ell k}$, that is the only deleted symbol that is equal to $k (\mathrm{mod} \ell)$, is $\ell \cdot s + k$, where $(s, s', s'') = \mathcal{D}^E_{n/\ell}(\frac{\sigma_{\sigma \equiv_\ell k} - k}{\ell})$. For simplicity of presentation, in the algorithm and in this discussion we ignore the possibility that $s' = 0$ or $s'' = 0$; these cases can be handled similarly.

Since the ℓ symbols of A are consecutive in π, there is one element of A that is equal to $k (\mathrm{mod} \ell)$. But since $\ell \cdot s + k$ is the only deleted element from π that is equal to k modulo ℓ, we have $\ell \cdot s + k \in A$. This in turn implies that the symbols of A are located between the symbol $\ell \cdot s' + k$ and the symbol $\ell \cdot s'' + k$ in π. So the size of the set $\mathcal{S}$ in step 3 is at most $(2\ell - 1) - \ell = \ell - 1$, where $2\ell - 1$ is the number of symbols between $\ell \cdot s' + k$ and $\ell \cdot s'' + k$ in π and ℓ is the number of symbols in A. Hence, $|\sigma^{(2)}| = n - |\mathcal{I}| - |\mathcal{S}| \leq n - (3\ell - 2)$.

In step 4 of the algorithm, $\ell \cdot s + k$ is inserted in its correct position. So now there are only $3\ell - 3$ elements missing from $\sigma^{(3)}$ compared to π. In other words, there is a set $\mathcal{I}'$ such that $\sigma^{(3)} = D(\pi, \mathcal{I}')$, where $|\mathcal{I}'| \leq 3\ell - 3$. Since $\ell \cdot s + k \notin \mathcal{I}'$, we have $b(\pi, \mathcal{I}', \ell) = 0$. Hence, by Claim 10, $\sigma^{(4)} = Er(\pi, \mathcal{I}')$ at step 6. The decoder $\mathcal{D}^H_{3\ell-2,n}$ can recover π from $\sigma^{(4)}$ in step 7 since $|\mathcal{I}'| \leq 3\ell - 3$ and the minimum Hamming distance of the code $C^U_{2\ell,n}$ is $3\ell - 2$. □

We illustrate the decoder $\mathcal{D}^U_{2\ell,n}$ with the following example.

Example 9. *Suppose* $\pi = (9, 7, \boldsymbol{8}, \boldsymbol{6}, \boldsymbol{1}, 11, 3, \boldsymbol{10}, 2, 12, \boldsymbol{4}, 5) \in \mathcal{C}^U_{6,12}$ *so that* $\ell = 3$. *Suppose* $\sigma = D(\pi, \mathcal{I}) = (9, 7, 11, 3, 2, 12, 5)$ *where* $\mathcal{I} = \{1, 4, 6, 8, 10\}$. *We will show that, if* $\mathcal{D}^U_{2\ell,n}$ *is used with* σ *as input, then* $\hat{\pi} = \pi$.

At step 1, we have $\sigma^{(1)} = (9, 7, 11, 3, ?, 2, 12, ?, 5)$. *Then since* $|\sigma^{(1)}| = 9 < 12$, *we proceed to step 2. At step 2,* $k = 0$ *since* $|\sigma_{\sigma \equiv 3 0}| = |(9, 3, 12)| = 4 - 1$. *Then at step 3, we the output of* $\mathcal{D}^E_4$ *would be* $(s, s', s'') = (2, 3, 1)$. *At step 3, the set* $\mathcal{S} = \sigma[9, 3] = \{7, 11\}$. *Notice that* $|\mathcal{S}| = 2$ *in this case. After the elements from the set* $\mathcal{S}$ *are removed from* σ, *at step 5 we have* $\sigma^{(2)} = (9, 3, 2, 12, 5)$. *At step 4, we insert the symbol* 6 *into* $\sigma^{(2)}$ *giving that*

$\sigma^{(3)} = (9, 6, 3, 2, 12, 5)$. *At step 6,* $\sigma^{(4)} = (9, ?, ?, 6, ?, ?, 3, ?, 2, 12, ?, 5)$. *Since there are* 6 *total erasures and* $\mathcal{D}^H_{7,12}$ *is the decoder for a code with Hamming distance* 7*, the output of* $\mathcal{D}^H_{7,12}$ *at step 7 will be* $\hat{\pi} = (9, 7, 8, 6, 1, 11, 3, 10, 2, 12, 4, 5) = \pi$ *as desired.*

We now briefly compare a code created according to Construction C to the codes from [FSM13]. A code $\mathcal{C}^U_{2\ell,n}$ with Ulam distance 2ℓ requires interleaving ℓ subsequences as a consequence of item 1) in Construction C. We note that, if the codes from [FSM13] were adopted, then constructing a code with Ulam distance 2ℓ would require interleaving at least $2(\ell - 1) + 1$ subsequences. Because Construction C requires the interleaving of fewer subsequences, it can be shown that for large n Construction C produces codebooks with much higher cardinalities than the codes presented in [FSM13]. In the next section, we consider codes that correct unstable deletions.

4.3 Properties of an Unstable Deletion

In this section, we begin with some useful notation that will be used to describe an unstable deletion and an unstable insertion. Suppose $\sigma \in \mathbb{S}_{n-1}$ is the result of an unstable deletion occurring to the permutation $\pi \in \mathbb{S}_n$ of the symbol $s \in [n]$. Then we write $\sigma = \pi_{\downarrow,s}$. If $\sigma \in \mathbb{S}_{n+1}$ is the result of an unstable insertion of the symbol $s \in [n+1]$ at position $j \in [n+1]$ into the permutation $\pi \in \mathbb{S}_n$, then we write $\sigma = \pi^{\uparrow,s,j}$.

For a permutation $\pi \in \mathbb{S}_n$, let $\mathcal{B}_D(\pi)$ be the set of all permutations given that a single deletion occurred to π, i.e., $\mathcal{B}_D(\pi) = \{\pi_{\downarrow,s} \ : \ s \in [n]\}$. We say that a code $\mathcal{C}$ is a ***single-deletion-correcting code*** if for any $\pi, \sigma \in \mathcal{C}$, $\mathcal{B}_D(\pi) \cap \mathcal{B}_D(\sigma) = \emptyset$. Similarly, $\mathcal{B}_I(\pi)$ is the set of all permutations given that a single insertion occurred to π, that is, $\mathcal{B}_I(\pi) = \{\pi^{\uparrow,s,j} \ : \ s, j \in [n+1]\}$. We say that a code $\mathcal{C}$ is a ***single-insertion-correcting code*** if for any $\pi, \sigma \in \mathcal{C}$, $\mathcal{B}_I(\pi) \cap \mathcal{B}_I(\sigma) = \emptyset$.

We first begin with a few basic properties that are consequences of Definitions 10.

Claim 11. *Let* $\pi = (\pi_1, \ldots, \pi_n) \in \mathbb{S}_n$*. Then, for any* $j \in [n]$, $\pi = (\pi_{\downarrow,\pi_j})^{\uparrow,\pi_j,j}$*. Similarly for*

any $s, j \in [n+1]$ $\pi = (\pi^{\uparrow,s,j})_{\downarrow,s}$.

Claim 12. *Let* $\pi = (\pi_1, \ldots, \pi_n) \in \mathbb{S}_n$, $s, t \in [n]$ *and suppose that* $s < t$. *Then* $(\pi_{\downarrow,s})_{\downarrow,t-1} = (\pi_{\downarrow,t})_{\downarrow,s}$.

Claim 13. *Let* $\pi \in \mathbb{S}_n$ *and* $j, k, s, t \in [n+1]$ *such that* $s \leq t$. *If* $j < k$, *then* $(\pi^{\uparrow,s,j})^{\uparrow,t+1,k+1} = (\pi^{\uparrow,t,k})^{\uparrow,s,j}$. *If* $j > k$, *then* $(\pi^{\uparrow,s,j})^{\uparrow,t+1,k} = (\pi^{\uparrow,t,k})^{\uparrow,s,j+1}$. *If* $j = k$ *and* $s < t$, *then* $(\pi^{\uparrow,s,j})^{\uparrow,t+1,j} = (\pi^{\uparrow,t,j})^{\uparrow,s,j+1}$.

The last three claims will be useful in showing that, similarly to the traditional setup of deletions and insertions in vectors [LEV66], there is also a duality between insertions and deletions in permutations.

Lemma 23. *For any* $\pi, \sigma \in \mathbb{S}_n$, $\mathcal{B}_D(\pi) \cap \mathcal{B}_D(\sigma) \neq \emptyset$ *if and only if* $\mathcal{B}_I(\pi) \cap \mathcal{B}_I(\sigma) \neq \emptyset$.

Proof. We first prove that if $\mathcal{B}_D(\pi) \cap \mathcal{B}_D(\sigma) \neq \emptyset$ then $\mathcal{B}_I(\pi) \cap \mathcal{B}_I(\sigma) \neq \emptyset$. Let $\tau \in \mathcal{B}_D(\pi) \cap \mathcal{B}_D(\sigma)$ so there exist $j, k \in [n]$ such that $\pi_{\downarrow,\pi_j} = \sigma_{\downarrow,\sigma_k} = \tau$. Notice that from Claim 11, $\pi = \tau^{\uparrow,\pi_j,j}$ and $\sigma = \tau^{\uparrow,\sigma_k,k}$. Assume without loss of generality that $\pi_j \leq \sigma_k$. We first consider the case where $j < k$. From Claim 13, we conclude that

$$\pi^{\uparrow,\sigma_k+1,k+1} = (\tau^{\uparrow,\pi_j,j})^{\uparrow,\sigma_k+1,k+1} = (\tau^{\uparrow,\sigma_k,k})^{\uparrow,\pi_j,j} = \sigma^{\uparrow,\pi_j,j},$$

and thus $\mathcal{B}_I(\pi) \cap \mathcal{B}_I(\sigma) \neq \emptyset$. Suppose $j > k$ where, as before, $\pi_j \leq \sigma_k$. From Claim 13, we have, $(\tau^{\uparrow,\pi_j,j})^{\uparrow,\sigma_k+1,k} = (\tau^{\uparrow,\sigma_k,k})^{\uparrow,\pi_j,j+1}$ and so $\mathcal{B}_I(\pi) \cap \mathcal{B}_I(\sigma) \neq \emptyset$ in this case as well. Now suppose $j = k$. First notice that if $j = k$ and $\pi_j = \sigma_k$, then $\pi = \sigma$ and so the result trivially holds. Therefore we assume $\pi_j < \sigma_k$. Then from Claim 13, we have $(\tau^{\uparrow,\pi_j,j})^{\uparrow,\sigma_k+1,j} = (\tau^{\uparrow,\sigma_k,j})^{\uparrow,\pi_j,j+1}$ and again $\mathcal{B}_I(\pi) \cap \mathcal{B}_I(\sigma) \neq \emptyset$.

Now we prove that if $\mathcal{B}_I(\pi) \cap \mathcal{B}_I(\sigma) \neq \emptyset$ then $\mathcal{B}_D(\pi) \cap \mathcal{B}_D(\sigma) \neq \emptyset$. Let $\theta \in \mathcal{B}_I(\pi) \cap \mathcal{B}_I(\sigma)$ and thus there exists $j, k, s, t \in [n+1]$ such that

$$\pi^{\uparrow,s,j} = \sigma^{\uparrow,t,k} = \theta = (\theta_1, \ldots, \theta_{n+1}).$$

Recall that from Claim 11, we have $\pi = \theta_{\downarrow,s}$ and $\sigma = \theta_{\downarrow,t}$. Notice that if $s = t$, then $j = k$ by Definition 10. Furthermore, if $j = k$ and $s = t$ then $\pi = \sigma$ and the result is straightforward. Suppose, without loss of generality, that $s < t$. From Claim 12,

$$\pi_{\downarrow,t-1} = (\theta_{\downarrow,s})_{\downarrow,t-1} = (\theta_{\downarrow,t})_{\downarrow,s} = \sigma_{\downarrow,s}.$$

By the assumption $s < t$, we have $s, (t-1) \in [n]$, and thus we showed that $\mathcal{B}_D(\pi) \cap \mathcal{B}_D(\sigma) \neq \emptyset$, as required. □

The following corollary follows directly from Lemma 23.

Corollary 8. *A code $\mathcal{C} \subseteq \mathbb{S}_n$ is a single-deletion-correcting code if and only if it is a single-insertion-correcting code.*

The following definition will be useful for characterizing permutations in the next section.

Definition 12. *For a permutation $\pi = (\pi_1, \ldots, \pi_n) \in \mathbb{S}_n$, a **consecutive run** is a substring of maximal length in π that contains consecutively valued symbols, increasing or decreasing.*

For example, if $\pi = (1, 5, 4, 3, 2) \in \mathbb{S}_5$, then π has 2 consecutive runs: (1) and $(5, 4, 3, 2)$.

4.4 An Upper Bound on the Cardinality of Single UD Codes

The goal of this section is to derive an upper bound on the maximum size of a single-deletion-correcting code. We refer to the maximum size of such codes over $\mathbb{S}_n$ by $A(n)$ and our upper bound on $A(n)$ will be given in Theorem 10.

We first note that the size of the deletion ball $\mathcal{B}_D(\pi)$ depends on the permutation π. However, as will be shown in the following lemma, the size of the deletion ball for a permutation π can be solely characterized as a function of the number of consecutive runs in π. For a vector, $\boldsymbol{v} = (v_1, \ldots, v_n)$ and two integers i_1, i_2, we denote by $\boldsymbol{v}_{[i_1,i_2]}$ the vector $\boldsymbol{v}_{[i_1,i_2]} = (v_{i_1}, \ldots, v_{i_2})$. As a formality, if $i_2 < i_1$, then $\boldsymbol{v}_{[i_1,i_2]}$ is the empty string.

Lemma 24. *Let $\pi = (\pi_1, \ldots, \pi_n) \in \mathbb{S}_n$ and suppose the symbols π_j and π_k ($j, k \in [n]$) belong to the same consecutive run in π. Then, $\pi_{\downarrow,\pi_j} = \pi_{\downarrow,\pi_k}$.*

Proof. We denote $\sigma = \pi_{\downarrow,\pi_j}$ and $\theta = \pi_{\downarrow,\pi_k}$ and assume without loss of generality that $j < k$. Let $(\pi_j, \pi_{j+1}, \ldots, \pi_k)$ be a substring of the consecutive run shared by the symbols π_j, π_k. Assume that $\pi_k > \pi_j$, while the opposite case is proven similarly.

Since π_j and π_k are in the same consecutive run in π, then from Definition 10, $\sigma_{[1,j-1]} = \theta_{[1,j-1]}$ and $\sigma_{[k,n-1]} = \theta_{[k,n-1]}$. Since $(\pi_j, \ldots, \pi_k)$ is a substring of an increasing consecutive run, then $\pi_{j+1} = \pi_j + 1, \pi_{j+2} = \pi_{j+1} + 1, \ldots, \pi_k = \pi_{k-1} + 1$. If π_k is deleted from $(\pi_j, \ldots, \pi_k)$ then the resulting substring is $(\pi_j, \ldots, \pi_{k-1})$. If π_j is deleted from $(\pi_j, \pi_{j+1}, \ldots, \pi_k)$, then the resulting substring is $(\pi_{j+1} - 1, \pi_{j+2} - 1, \ldots, \pi_k - 1) = (\pi_j, \ldots, \pi_{k-1})$, and therefore $\sigma_{[j,k-1]} = \theta_{[j,k-1]}$. Thus, we conclude that $\sigma = \theta$. □

The converse of Lemma 24 is proven next.

Lemma 25. *Let $\pi = (\pi_1, \ldots, \pi_n) \in \mathbb{S}_n, j, k \in [n]$. If $\pi_{\downarrow,\pi_j} = \pi_{\downarrow,\pi_k}$ then π_j and π_k belong to the same consecutive run.*

Proof. Assume without loss of generality, that $j < k$. Let us denote

$$\pi_{\downarrow,\pi_j} = \pi_{\downarrow,\pi_k} = \pi' = (\pi'_1, \ldots, \pi'_{n-1}). \tag{4.1}$$

The proof will be by induction on $(k - j)$. For the base case, we prove that the result holds when $(k - j) = 1$. Since $\pi' = \pi_{\downarrow,\pi_j}$, we have that $\pi'_j = \pi_{j+1} - 1$ if $\pi_j < \pi_{j+1}$ or $\pi'_j = \pi_{j+1}$ if $\pi_j > \pi_{j+1}$. Similarly, since $\pi' = \pi_{\downarrow,\pi_k}$, we have that $\pi'_j = \pi_j$ if $\pi_j < \pi_k$ or $\pi'_j = \pi_j - 1$ if $\pi_j > \pi_k$. Thus, since $\pi \in \mathbb{S}_n$, we have that either $\pi'_j = \pi_j = \pi_{j+1} - 1$ if $\pi_j < \pi_k$ or $\pi'_j = \pi_j - 1 = \pi_{j+1}$ otherwise. In either case, (π_j, π_{j+1}) is a substring of a consecutive run.

Suppose that for all $(k - j) < m$, if $\pi_{\downarrow,\pi_j} = \pi_{\downarrow,\pi_k}$ then π_j and π_k belong to the same consecutive run and consider the case where $(k - j) = m$ and $\pi_{\downarrow,\pi_j} = \pi_{\downarrow,\pi_k}$. First, we show that if $\pi_{\downarrow,\pi_j} = \pi_{\downarrow,\pi_k}$, then $\pi_{\downarrow,\pi_j} = \pi_{\downarrow,\pi_{j+1}}$. Since assumption (4.1) holds, then as before either

1) $\pi'_j = \pi_{j+1}$, or 2) $\pi'_j = \pi_{j+1} - 1$. Using the same logic, we can conclude that (π_j, π_{j+1}) is a substring of a consecutive run and from Lemma 24, it follows that $\pi_{\downarrow,\pi_j} = \pi_{\downarrow,\pi_{j+1}} = \pi_{\downarrow,\pi_k}$.

Let $\ell = j + 1$. Since $(k - \ell) < m$, we can use the inductive hypothesis to conclude that since $\pi_{\downarrow,\pi_k} = \pi_{\downarrow,\pi_\ell}$, then π_k, π_ℓ are in the same consecutive run. Since π_j, π_ℓ are in the same consecutive run and π_ℓ, π_k are in the same consecutive run, it follows that π_j, π_k are in the same consecutive run and the proof is complete. □

For a permutation $\pi \in \mathbb{S}_n$, we denote by $R(\pi)$ the number of consecutive runs in π. The following is a corollary of Lemmas 24 and 25.

Corollary 9. *For all $\pi \in \mathbb{S}_n$, $|\mathcal{B}_D(\pi)| = R(\pi)$.*

We introduce some terminology that will be used for the derivation of the upper bound on $A(n)$. For positive integers n, r where $r < n$, we define

$$F(n, r) = \begin{pmatrix} n-1 \\ r-1 \end{pmatrix} \cdot 2^{\min\{r, n-r\}} \cdot r!. \tag{4.2}$$

To simplify the notation, we assume that n is a power of two so that the floors and ceilings can be dropped for convenience. We also assume that all log functions are base 2.

Lemma 26. *The number of permutations from $\mathbb{S}_n$ with r $(1 \leq r \leq n)$ consecutive runs is at most $F(n, r)$.*

Proof. Consider the set of permutations from $\mathbb{S}_n$ that contain r consecutive runs. We proceed by over-counting this quantity. We first partition the elements from $[n]$ into r consecutive runs. This is equivalent to computing the number of solutions to the problem $\sum_{j=1}^{r} t_j = n$, where $t_j \geq 1$ and each t_j is an integer. There are $\begin{pmatrix} n-1 \\ r-1 \end{pmatrix}$ such solutions.

If $r \leq \frac{n}{2}$ then there can be at most r consecutive runs of length greater than one. Each consecutive run can be either increasing or decreasing and so there are at most 2^r ways to

re-arrange the numbers within each consecutive run. If $r > \frac{n}{2}$ then there are at most $n - r$ consecutive runs of length greater than one. In this case there are 2^{n-r} ways to re-arrange the numbers within each consecutive run. Then, if we permute each (block of symbols that constitute each) consecutive run we have at most

$$\left(\begin{array}{c} n-1 \\ r-1 \end{array} \right) \cdot 2^{\min\{r,n-r\}} \cdot r!$$

permutations in $\mathbb{S}_n$ with r consecutive runs. □

We need the following claim and lemma in order to prove Theorem 10.

Claim 14. *For $2 \leq r \leq n - \log(n)$, $F(n, r-1) \leq F(n, r)$.*

Lemma 27. *The number of permutations in $\mathbb{S}_n$ with at most $n - \log(n)$ consecutive runs is at most $\frac{n!(n-\log(n))^2}{(\log(n))!}$.*

Proof. From Claim 14, the maximum number of permutations in $\mathbb{S}_n$ with fewer than $n - \log(n)$ consecutive runs is at most $\sum_{r=1}^{n-\log(n)} F(n, r) \leq \sum_{r=1}^{n-\log(n)} F(n, n-\log(n))$. Substituting the expression for $F(n, n - \log(n))$ from (4.2) gives that there are at most

$$\begin{aligned} &(n - \log(n)) \cdot \left(\begin{array}{c} n-1 \\ n-\log(n)-1 \end{array} \right) \cdot 2^{\log(n)} \cdot (n - \log(n))! \\ &= \frac{n!(n - \log(n))^2}{(\log(n))!} \end{aligned}$$

permutations in $\mathbb{S}_n$ with at most $n - \log(n)$ consecutive runs. □

Using a similar approach as in [LEV66], we provide an upper bound for the maximum size of a single-deletion-correcting code.

Theorem 8. *For any $0 < \epsilon < 1$ there exists an N_ϵ such that for all $n \geq N_\epsilon$, $A(n) \leq \frac{n!}{n(n-\log(n))}(1 + \epsilon)$.*

Proof. Suppose $\mathcal{C}$ is a single-deletion-correcting code over $\mathbb{S}_n$. Let $\mathbb{S}'_n = \{\pi' \in \mathbb{S}_n : |\mathcal{B}_D(\pi')| > n - \log(n)\}$. We first consider an upper bound on $|\mathcal{C} \cap \mathbb{S}'_n|$. Since the sets $\mathcal{B}_D(\pi') \subseteq \mathbb{S}_{n-1}$, for all $\pi' \in \mathcal{C} \cap \mathbb{S}'_n$ are disjoint and $|\mathcal{B}_D(\pi')| > n - \log(n)$ we get,

$$|\mathcal{C} \cap \mathbb{S}'_n| \leq \frac{(n-1)!}{n - \log(n)}.$$

Let $\mathbb{S}''_n = \{\pi'' \in \mathbb{S}_n : |\mathcal{B}_D(\pi'')| \leq n - \log(n)\}$. Clearly, $|\mathcal{C} \cap \mathbb{S}''_n| \leq |\mathbb{S}''_n|$ and from Lemma 27, $|\mathbb{S}''_n| \leq \frac{n!(n-\log(n))^2}{(\log(n))!}$. Thus, we have

$$\begin{aligned} |\mathcal{C}| =& |\mathcal{C} \cap \mathbb{S}'_n| + |\mathcal{C} \cap \mathbb{S}''_n| \leq \frac{(n-1)!}{n - \log(n)} + \frac{n!(n - \log(n))^2}{(\log(n))!} \\ &= \frac{n!}{n(n - \log(n))} \left(1 + \frac{n(n - \log(n))^3}{(\log(n))!}\right). \end{aligned}$$

Since this upper bound holds for any single-deletion-correcting code $\mathcal{C}$, we conclude that $A(n) \leq \frac{n!}{n(n-\log(n))}\left(1+\frac{n(n-\log(n))^3}{(\log(n))!}\right)$. Lastly, since $\lim_{n\to\infty} \frac{n(n-\log(n))^3}{(\log(n))!} = 0$, there exists an N_ϵ such that for all $n \geq N_\epsilon$, we have $A(n) \leq \frac{n!}{n(n-\log(n))}(1+\epsilon)$. □

4.5 Single UD Code Construction

In this section, we first consider some properties regarding deletions in permutations. Afterwards, an asymptotically optimal construction of single-deletion-correcting codes is provided. Lastly, we prove the correctness of the construction and discuss a decoding algorithm.

4.5.1 Permutation Matrices, Signatures, and Runs

A ***permutation matrix*** is a square binary matrix such that in each row and each column there is precisely one 1. For a permutation $\pi = (\pi_1, \ldots, \pi_n) \in \mathbb{S}_n$, the ***permutation matrix of*** π is a permutation matrix $M = f(\pi)$ such that $M_{ij} = 1$ if $j = \pi_i$. Notice that if M is an $n \times n$ permutation matrix, then there exists an unique permutation π such that $M = f(\pi)$. Hence, the mapping f is invertible. We denote its inverse by f^{-1} and write $f^{-1}(f(\pi)) = \pi$.

The next claim follows from Definition 10.

Claim 15. *For $\pi \in \mathbb{S}_n$, the matrix $f(\pi_{\downarrow,\pi_j})$ is the result of removing row j and column π_j from $f(\pi)$. Furthermore, if row j and column π_j are removed from $f(\pi)$ thus resulting in M', then $\pi_{\downarrow,\pi_j} = f^{-1}(M')$.*

We next use a mapping, referred to as the ***signature***, that will be useful in our code construction. This mapping was also used in [TEN84] in the context of single-deletion-correcting codes over non-binary vectors. Let $m > 2$ be an integer. For $\boldsymbol{y} \in [m]^n$, define the binary length-$(n-1)$ signature $\alpha(\boldsymbol{y}) = (\alpha(\boldsymbol{y})_1, \ldots, \alpha(\boldsymbol{y})_{n-1})$ as follows. For $1 \leq i \leq n-1$,

$$\alpha(\boldsymbol{y})_i = \begin{cases} 1, & \text{if } y_{i+1} \geq y_i. \\ 0, & \text{otherwise.} \end{cases} \tag{4.3}$$

For a permutation $\pi = (\pi_1, \ldots, \pi_n) \in \mathbb{S}_n$, recall from [ROS04] that the inverse permutation $\pi^{-1} = (\pi_1^{-1}, \ldots, \pi_n^{-1}) \in \mathbb{S}_n$ is such that for $i \in [n]$, π_i^{-1} is the location of the element i in π. It is straightforward to verify that the permutation of the transpose of $f(\pi)$ corresponds to the inverse permutation for π. In other words, we have $f(\pi^{-1}) = (f(\pi))^T$. For shorthand, we will refer to the signature of the inverse permutation as the ***inverse signature***. The next example illustrates a signature and an inverse signature.

Example 10. *Suppose $\pi = (1,3,5,4,2) \in \mathbb{S}_5$. Then $\alpha(\pi) = (1,1,0,0)$. Furthermore, $\pi^{-1} = (1,5,2,4,3) \in \mathbb{S}_5$ and $\alpha(\pi^{-1}) = (1,0,1,0)$.*

We are now ready to prove the following lemma.

Lemma 28. *For $\pi \in \mathbb{S}_n$, we have $(\pi_{\downarrow,\pi_j})^{-1} = (\pi^{-1})_{\downarrow,j}$.*

Proof. From Claim 15, if the symbol π_j (where $j \in [n]$) is deleted from π, then the permutation matrix $f(\pi_{\downarrow,\pi_j})$ is the result of removing row j and column π_j from $f(\pi)$. Alternatively, we can obtain $(f(\pi_{\downarrow,\pi_j}))^T$ by removing row π_j and column j from $(f(\pi))^T$. From Claim 15,

removing row π_j and column j from $(f(\pi))^T$ corresponds to the deletion of symbol j from π^{-1} since $f^{-1}((f(\pi))^T) = \pi^{-1}$. □

In the following, if the input to $\mathcal{B}_D$ is a binary vector $\boldsymbol{x} \in GF(2)^n$, then the output of $\mathcal{B}_D(\boldsymbol{x})$ is the set of all possible vectors obtainable by deleting one bit from $\boldsymbol{x}$. If a symbol from a permutation π is deleted, then a symbol from its signature, $\alpha(\pi)$, is deleted as well. That is, $\alpha(\pi_{\downarrow,x}) \in \mathcal{B}_D(\alpha(\pi))$. Hence, an immediate consequence of Lemma 28 is the following.

Corollary 10. *Let $\pi \in \mathbb{S}_n$ and $j \in [n]$. Then $\alpha(\pi_{\downarrow,\pi_j}) \in \mathcal{B}_D(\alpha(\pi))$ and $\alpha((\pi_{\downarrow,\pi_j})^{-1}) \in \mathcal{B}_D(\alpha(\pi^{-1}))$.*

A binary code will be called a ***binary single-deletion-correcting code*** if it can correct any single-bit deletion. As a result of Corollary 10, in the next subsection we will leverage binary single-deletion-correcting codes which will be invoked over the signature and inverse signature of permutations in order to construct single-deletion-correcting codes for permutations.

In the rest of this subsection, we define runs in permutations and binary sequences and present a claim that will be useful in the next subsection.

Definition 13. *For a binary sequence s, a **run** is a maximal substring of s that is all-zero or all-one. For a permutation π, an **ascending run** is a maximal substring of π whose values are increasing and a **descending run** is a maximal substring of π whose values are decreasing. A substring of π is a **run** if it is an ascending or a descending run.*

Example 11. *Continuing with the setup from Example 10, let $\pi = (1, 3, 5, 4, 2) \in \mathbb{S}_5$ and $\alpha(\pi) = (1, 1, 0, 0)$. Notice that the signature of a permutation reflects the structure of the runs in the permutation. For example, the first three symbols in π comprise an ascending run and so the first two symbols of $\alpha(\pi)$ are ones.*

The following claim is straightforward to verify.

Claim 16. *Consider a permutation $\pi \in \mathbb{S}_n$ and its signature $\alpha(\pi)$. There is an ascending run starting at position i and ending at position $j+1$ in π if and only if there is a run of ones starting at position i and ending at position j in $\alpha(\pi)$. Furthermore, if $\sigma = \pi_{\downarrow,x}$, where x is at position k in π with $i \leq k \leq j+1$, then $\alpha(\pi)$ can be converted to $\alpha(\sigma)$ by deleting a 1 from this run. A similar statement holds for descending runs and deletion of 0s.*

4.5.2 Code construction

Let us first review the binary single-deletion-correcting code we will use in our construction. Namely, the code from [VT65], known as a ***Varshamov-Tennegolts code*** (VT code), is defined as follows. For a positive integer $n > 2$ and $a \in \mathbb{Z}_{n+1}$, $\mathcal{C}_a^n$ is the code $\mathcal{C}_a^n = \{\boldsymbol{x} \in GF(2)^n : \sum_{i=1}^n ix_i \equiv a \mod n+1\}$.

Lemma 29. *(cf. [LEV66]) For any integer $n > 2$ and $a \in \mathbb{Z}_{n+1}$, the code $\mathcal{C}_a^n$ is a binary single-deletion-correcting code.*

We are now ready to present our code construction of single-deletion-correcting codes over permutations.

Construction D. *Given an integer $n > 2$ and $a_1, a_2 \in \mathbb{Z}_n$, let*

$$\mathcal{C}_{a_1,a_2}^n = \left\{\pi \in \mathbb{S}_n : \alpha(\pi) \in \mathcal{C}_{a_1}^{n-1}, \alpha(\pi^{-1}) \in \mathcal{C}_{a_2}^{n-1}\right\}. \tag{4.4}$$

We first comment on the cardinality of the codes $\mathcal{C}_{a_1,a_2}^n$. Recall from the previous section that $A(n)$ represents the maximum cardinality of a single-deletion-correcting code. Note that the codes $\mathcal{C}_{a_1,a_2}^n$ for $a_1, a_2 \in \mathbb{Z}_n$ partition the space $\mathbb{S}_n$ into n^2 mutually disjoint codes. Hence, if we denote by $F(n)$ for $n > 2$ the maximum cardinality of a code according to Construction D, that is, $F(n) = \max_{a_1,a_2 \in \mathbb{Z}_n}\{|\mathcal{C}_{a_1,a_2}^n|\}$, then applying the pigeonhole principle gives the following result.

Corollary 11. *Construction D is asymptotically optimal, that is,*

$$\lim_{n\to\infty} \frac{F(n)}{A(n)} = 1.$$

4.5.3 Proof of Correctness

In this subsection, we prove the correctness of Construction D. From our proof, a decoding algorithm for the codes can be easily derived. Due to lack of space, however, we omit an explicit presentation of the decoding algorithm.

For simplicity, for a permutation σ, we use $x \prec_\sigma y$ if and only if $\sigma^{-1}(x) < \sigma^{-1}(y)$.

Theorem 9. *For $n > 2$ and $a_1, a_2 \in \mathbb{Z}_n$, the code $\mathcal{C}^n_{a_1,a_2}$ is a single-deletion-correcting permutation code.*

Proof. Suppose that $\pi \in \mathcal{C}^n_{a_1,a_2}$ and that $\sigma = \pi_{\downarrow,x}$ for some $x \in [n]$. We show that π is uniquely identifiable from σ. To do this, we first identify the runs in σ and σ^{-1} from which x was deleted and show that there is an unique way to increase the length of these runs by an insertion in a consistent way.

Let k denote the position of x in π, that is, we have $\pi = \sigma^{\uparrow,x,k}$. From the received word σ, we compute $\alpha(\sigma)$. Corollary 10 implies that $\alpha(\sigma) \in \mathcal{B}_D(\alpha(\pi))$. Using a decoder for a VT code, we find $\alpha(\pi)$ from $\alpha(\sigma)$ since there is a deletion that converts $\alpha(\pi)$ to $\alpha(\sigma)$. By comparing $\alpha(\pi)$ and $\alpha(\sigma)$, we find the i and j of Claim 16. Hence, $\pi_i, \pi_{i+1}, \ldots, \pi_{j+1}$ form a run in π. Without loss of generality, assume that this run is an ascending run, or equivalently, the deleted element in $\alpha(\pi)$ is a 1. Thus, by Claim 16, $\pi = \sigma^{\uparrow,x,k}$ such that

$$i \le k \le j+1, \tag{4.5}$$

$$\sigma_i < \sigma_{i+1} < \cdots < \sigma_j, \tag{4.6}$$

$$x \le \sigma_m \text{ iff } k \le m \text{ for } m \in \{i, \ldots, j\}. \tag{4.7}$$

By Lemma 28, we have $\sigma^{-1} = (\pi^{-1})_{\downarrow,k}$ and $\pi^{-1} = (\sigma^{-1})^{\uparrow,k,x}$. We thus may apply Claim 16

by substituting π with π^{-1}. Let the corresponding values of i and j of the claim be denoted by p and q, respectively, for this case. The claim implies that the substring $\pi_p^{-1}, \pi_{p+1}^{-1}, \ldots, k, \ldots, \pi_q^{-1}, \pi_{q+1}^{-1}$ is a run of π^{-1} and that $p \leq x \leq q+1$.

The values of p and q can be determined from $\alpha(\sigma^{-1})$ as follows. By Claim 16 and using a decoder for a VT code, we find $\alpha(\pi^{-1})$ from $\alpha(\sigma^{-1})$ since there is a deletion that converts $\alpha(\pi^{-1})$ to $\alpha(\sigma^{-1})$. We then find p and q by comparing $\alpha(\pi^{-1})$ and $\alpha(\sigma^{-1})$.

There are two different cases depending on the deleted element of $\alpha(\pi^{-1})$ being a 1 or a 0. Due to lack of space, here, we only consider the former case. Suppose that a 1 in a run of 1s in $\alpha(\pi^{-1})$ is deleted. We have $\pi_p^{-1} < \pi_{p+1}^{-1} < \cdots < k < \cdots < \pi_q^{-1} < \pi_{q+1}^{-1}$ and thus

$$x \in \{p, p+1, \ldots, q+1\}, \tag{4.8}$$

$$p \prec_\sigma p+1 \prec_\sigma \cdots \prec_\sigma q, \tag{4.9}$$

$$k \leq \sigma^{-1}(y) \text{ iff } x \leq y \text{ for } y \in \{p, \ldots, q\}, \tag{4.10}$$

where $p \prec_\sigma$ denotes that the symbol p appears before the symbol $p+1$ in the permutation σ. Let $A = \{\sigma_i, \ldots, \sigma_j\} \cap \{p, \ldots, q\}$. Suppose A is empty. Because $\sigma_i, \ldots, \sigma_j$ is an increasing run of σ and $p, p+1, \ldots, q$ in an increasing subsequence of σ, we have one of the following cases: a) $\sigma^{-1}(q) < i$; b) $\sigma^{-1}(p) > j$; or c) $\sigma^{-1}(z-1) < i$ and $\sigma^{-1}(z) > j$ for some $z \in \{p+1, \ldots, q\}$. Note that if cases a) and b) do not hold, then $q > p$ and so the set $\{p+1, \ldots, q\}$ used in case c) is nonempty.

We consider each case separately: a) From (4.5), we have $\sigma^{-1}(q) < k$, which using (4.10) implies that $x > q$. From (4.8), we find $x = q+1$. b) Similar to case a), from (4.5), (4.10), and (4.8), we have $x = p$. c) From (4.5) it follows that $\sigma^{-1}(z-1) < k \leq \sigma^{-1}(z)$. Using (4.10), we find that $x = z$. Hence, we can identify x if A is empty. Having identified x, we can find the unique position k in $\{i, \ldots, j+1\}$ that satisfies condition (4.7).

Now suppose A is nonempty and so $u = \min A$ and $v = \max A$ are well defined. From (4.6) and (4.9), it is not difficult to show that every integer between u and v is also in A,

i.e.,

$$A = \{u, u+1, \ldots, v\},$$

and that the elements of A form a consecutive run in σ. Based on (4.5)–(4.10), it is straightforward (but tedious) to see that the set of possible values for x is exactly $\{u, u+1, \ldots, v, v+1\}$ and that $k = \sigma^{-1}(x)$ if $u \leq x \leq v$ and $k = \sigma^{-1}(v)+1$ if $x = v+1$. Furthermore, with the aforementioned values for x and k, the resulting permutation $\sigma^{\uparrow,x,k}$ is the same; it a permutation in which the length of the consecutive run formed by the element of A is increased by 1. Thus π is determined uniquely.

□

We illustrate the preceding proof by the following example.

Example 12. *Consider the code $\mathcal{C}^8_{a_1,a_2}$, where $a_1 = 7$ and $a_2 = 0$. Suppose the stored codeword is $\pi = (7,4,5,6,8,2,1,3) \in \mathcal{C}^8_{a_1,a_2}$, and the retrieved permutation is $\sigma = \pi_{\downarrow,5} = (6,4,5,7,2,1,3)$. The decoder is given σ, a_1, and a_2, and from these it computes*

$$\alpha(\sigma) = (0,1,1,0,0,1),$$
$$\alpha(\pi) = (0,1,1,1,0,0,1),$$
$$\alpha(\sigma^{-1}) = (0,1,0,1,0,1),$$
$$\alpha(\pi^{-1}) = (0,1,0,1,1,0,1).$$

We thus have $i = 2$, $j = 4$, $p = 4$, and $q = 5$. Furthermore, $A = \{4,5,7\} \cap \{4,5\} = \{4,5\}$, implying that $x \in \{4,5,6\}$. Hence, the possible pairs of values for (x,k) are $(4,2)$, $(5,3)$, and $(6,4)$. Note that $\pi = \sigma^{\uparrow,4,2} = \sigma^{\uparrow,5,3} = \sigma^{\uparrow,6,4}$, and so the decoding is successful.

4.6 Conclusion

Recall from Chaper 1, that rank modulation systems can be used to improve the reliability of emerging storage systems by storing information using the rankings of cells rather

than using the absolute cell levels. In this chapter, we studied codes capable of correcting synchronization errors under the rank modulation setup. We showed that codes in the Ulam metric can recover from a large class of synchronization errors. Furthermore, we presented new codes in the Ulam metric that improved upon the state of the art and for the case of a single unstable deletion, we constructed an asymptotically optimal code.

CHAPTER 5

Correcting Grain-Errors in Magnetic Media

5.1 Introduction and Preliminaries

This chapter studies new bounds and constructions that are applicable granular media recording. In this section, we formally define the channel model and introduces the notation and tools used for the remainder of the chapter. Section 5.2 improves upon the existing upper bounds from [SR11]. Section 5.3 contains constructions for codes that correct grain-errors and a related type of error which we refer to as mineral-errors. Lower bounds on the cardinalities for some of these codes are then derived in Section 5.4. Section 5.5 revisits the general approach to correcting grain/mineral-errors from Section 5.3.2, and identifies additional codes for certain code lengths.

In this section, we describe in detail the structure of grain-errors. Afterwards, we introduce some key notation. Section 5.1.1 introduces the errors of interest. Section 5.1.2 reviews the tools which will be used for computing upper bounds. Section 5.1.3 briefly introduces some graph notation. Section 5.1.4 reviews some distance metrics and group codes that will be useful for constructing grain-error-correcting codes. Finally, Section 5.1.5 includes some Fourier analysis tools useful for computing lower bounds for grain-error codes.

5.1.1 Grain-errors and mineral-errors

In this subsection, we formally introduce the notation and the errors of interest that will be studied in this work. We consider the case where each grain contains either one or two bits of data. A grain-error causes the two bits in the same two-bit grain to either both be 0 or both be 1; the error operation can be interpreted as a ***smearing***. Following the setup of [MBK11], we assume that the first bit smears the second. The problem of interest is how to correct grain-errors when the locations and lengths of the grains are unknown to both the encoder and decoder.

Before continuing, we provide a formal definition of a t-grain-error. For a vector $\boldsymbol{x} \in GF(2)^n$, $wt(\boldsymbol{x})$ refers to the Hamming weight of $\boldsymbol{x}$ and $supp(\boldsymbol{x})$ denotes the set of indices of $\boldsymbol{x}$ with non-zero values.

Definition 14. *Let $t \geq 1$ be an integer. Suppose a vector $\boldsymbol{x} \in GF(2)^n$ was stored. Let $\boldsymbol{e_x} = (e_1, \ldots, e_n) \in GF(2)^n$, and suppose the vector $\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{e_x}$ was read. Then, we say that $\boldsymbol{e_x}$ is a* **t-grain-error for $\boldsymbol{x}$** *if the following holds:*

1. *$wt(\boldsymbol{e_x}) \leq t$ and $e_1 = 0$,*
2. *For $2 \leq i \leq n$, if $e_i \neq 0$, then $x_i \neq x_{i-1}$.*

Note that $\boldsymbol{e_x}$ depends on the input vector $\boldsymbol{x}$. For shorthand, we say that $\boldsymbol{e_x}$ is a t-grain-error if the vector $\boldsymbol{x}$ is clear from the context. Notice in Definition 14 that an error at position i where $2 \leq i \leq n$ can be interpreted as a smearing where the value of $\boldsymbol{x}$ at position $i-1$ smears the value of $\boldsymbol{x}$ in position i.

A code that can correct any t-grain-error will be referred to as a ***t-grain-error-correcting code***. For shorthand, a code that can correct a single grain-error will also be referred to as a ***single-grain code***. More generally, codes that correct a prescribed number of grain-errors are called ***grain codes***. The maximum size of a t-grain-error-correcting-code of length n will be referred to as $M(n, t)$.

Definition 14 coincides with the ***overlapping grain-error*** model discussed in [SR11]. We briefly note that since the original model of ***non-overlapping grain-errors*** [MBK11] is a special case of the more general overlapping grain-error model, the code constructions in this chapter apply to both models. We compare the upper bounds derived in Section 5.2 against existing bounds for the overlapping grain-error model ([SR11]). For the remainder of the chapter, the term grain-error refers to an overlapping grain-error as stated in Definition 14.

Suppose a vector $\boldsymbol{x} \in GF(2)^n$ is stored. Let $\mathcal{B}_{t,G}(\boldsymbol{x})$ be the set of all possible vectors received (the ***error-ball***) given that any t-grain-error may occur in $\boldsymbol{x}$. That is, we define

$$\mathcal{B}_{t,G}(\boldsymbol{x}) = \{\boldsymbol{x} + \boldsymbol{e}_{\boldsymbol{x}} |\ \boldsymbol{e}_{\boldsymbol{x}} \text{ is a } t\text{-grain-error}\},$$

and $b_{t,n}(\boldsymbol{x}) = |\mathcal{B}_{t,G}(\boldsymbol{x})|$.

Example 13. *Suppose* $\boldsymbol{x} = (0,0,0,1,0)$ *was stored. Then,* $\mathcal{B}_{1,G}(\boldsymbol{x}) = \{(0,0,0,1,0), (0,0,0,1,1), (0,0,0,0,0)\}$ *and* $b_{1,5}(\boldsymbol{x}) = 3$. *Notice also that* $\mathcal{B}_{2,G}(\boldsymbol{x}) = \{(0,0,0,1,0), (0,0,0,1,1), (0,0,0,0,0), (0,0,0,0,1)\}$ *and* $b_{2,5}(\boldsymbol{x}) = 4$.

We note that the last vector, $(0,0,0,0,1)$, enumerated in $\mathcal{B}_{2,G}(\boldsymbol{x})$ for Example 13 was an overlapping grain-error in the sense that the grain-errors were adjacent so that the bit in position 4 is both smeared and smearing.

We introduce a new type of error that will be useful in subsequent analysis.

Definition 15. *Let* $t \geq 1$ *be an integer. Suppose a vector* $\boldsymbol{x} \in GF(2)^n$ *was stored. Let* $\boldsymbol{e}_{\boldsymbol{x}} = (e_1, \ldots, e_n) \in GF(2)^n$ *and suppose the vector* $\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{e}_{\boldsymbol{x}}$ *was received. Then, we say that* $\boldsymbol{e}_{\boldsymbol{x}}$ *is a* ***t*-mineral-error for** $\boldsymbol{x}$ *if the following holds:*

1. $wt(\boldsymbol{e}_{\boldsymbol{x}}) \leq t$,
2. *For* $2 \leq i \leq n$, *if* $e_i \neq 0$, *then* $x_i \neq x_{i-1}$.

Similar to the grain-error setup, we say that $\boldsymbol{e_x}$ is a t-mineral-error if the vector $\boldsymbol{x}$ is clear from the context. A code that can correct any t-mineral-error will be referred to as a ***t-mineral-error-correcting code***. ***Single-mineral codes*** and ***mineral codes*** are defined analogously as grain codes.

For a given vector $\boldsymbol{x} \in GF(2)^n$, let $\mathcal{B}_{t,M}(\boldsymbol{x})$ denote the error-ball for $\boldsymbol{x}$ given that any t-mineral-error may occur in $\boldsymbol{x}$. That is, we define

$$\mathcal{B}_{t,M}(\boldsymbol{x}) = \{\boldsymbol{x} + \boldsymbol{e_x} |\ \boldsymbol{e_x} \text{ is a } t\text{-mineral-error}\}.$$

A useful consequence of Definition 15 is stated in the following claim.

Claim 17. *Suppose $\mathcal{C}$ is a t-grain-error-correcting code. Then, for any two distinct codewords $\boldsymbol{x} = (x_1, \ldots, x_n), \boldsymbol{y} = (y_1, \ldots, y_n) \in \mathcal{C}$, either*

1. *$x_1 \neq y_1$, or*
2. *$\mathcal{B}_{t,M}(x_2, \ldots, x_n) \cap \mathcal{B}_{t,M}(y_2, \ldots, y_n) = \emptyset$.*

Suppose $\boldsymbol{x} \in GF(2)^n$ and $\mathcal{B}_{t,R}$ denotes the error-ball for ***t random-errors*** (where t random-errors are defined as any binary vector of length n with weight at most t). Then, for any vector $\boldsymbol{x} \in GF(2)^n$, $|\mathcal{B}_{t,R}(\boldsymbol{x})| = \sum_{i=0}^{t} \binom{n}{i}$.

The following lemma follows from the definitions of grain-errors and mineral-errors.

Claim 18. *For any vector $\boldsymbol{x} \in GF(2)^n$, $\mathcal{B}_{t,G}(\boldsymbol{x}) \subseteq \mathcal{B}_{t,M}(\boldsymbol{x}) \subseteq \mathcal{B}_{t,R}(\boldsymbol{x})$.*

We now present some simple results that follow from the structure of grain-errors. Lemmas 30, 31, and 33 will be useful in Section 5.2 for obtaining upper bounds on the cardinality of grain codes and Lemma 32 and Claim 19 will be useful for constructing grain codes in Section 5.3.

A ***run*** is a maximal substring of one or more consecutive identical symbols. We denote the number of runs in a vector $\boldsymbol{x}$ as $r(\boldsymbol{x})$ where $\boldsymbol{x} \in GF(2)^n$.

Lemma 30. *For any vector* $\boldsymbol{x}$, $b_{t,n}(\boldsymbol{x}) = \sum_{j=0}^{\min\{t,r(\boldsymbol{x})-1\}} \begin{pmatrix} r(\boldsymbol{x})-1 \\ j \end{pmatrix}$.

Proof. Suppose a vector $\boldsymbol{x}$ was stored and that it consists of $k = r(\boldsymbol{x})$ runs. By Definition 1, a grain-error can occur only at the boundaries between runs. If there are exactly $k \geq t+1$ runs, there are $k-1$ transitions between runs and therefore $b_{t,n}(\boldsymbol{x}) = \sum_{j=0}^{t} \begin{pmatrix} k-1 \\ j \end{pmatrix}$. If there are t or fewer runs (i.e., $k \leq t$), then $b_{t,n}(\boldsymbol{x}) = \sum_{j=0}^{k-1} \begin{pmatrix} k-1 \\ j \end{pmatrix}$. □

The following lemma is a consequence of the smearing effect of a grain-error. Let the map $\Psi : GF(2)^s \to GF(2)^{s-1}$ be defined so that $\Psi(\boldsymbol{z}) = \boldsymbol{z}' = (z'_1, \ldots, z'_{s-1})$ where $z'_i = (z_i + z_{i+1}) \bmod 2$ (for $1 \leq i \leq s-1$). Notice that $\Psi(\boldsymbol{z})$ is a linear map and it has a 1 in position i if and only if $z_i \neq z_{i+1}$. Recall that $supp(\boldsymbol{z})$ refers to the set of non-zero indices in $\boldsymbol{z}$ and $wt(\boldsymbol{z})$ refers to the Hamming weight of $\boldsymbol{z}$.

Lemma 31. *For any two vectors* $\boldsymbol{x}, \boldsymbol{y} \in GF(2)^n$ *if* $\boldsymbol{y} \in \mathcal{B}_{t,G}(\boldsymbol{x})$, *then* $r(\boldsymbol{y}) \leq r(\boldsymbol{x})$ *and* $b_{t,n}(\boldsymbol{y}) \leq b_{t,n}(\boldsymbol{x})$.

Proof. For the result to hold, we need to show that for any two vectors $\boldsymbol{x}, \boldsymbol{y} \in GF(2)^n$ where $\boldsymbol{y} \in \mathcal{B}_{t,G}(\boldsymbol{x})$, $r(\boldsymbol{y}) \leq r(\boldsymbol{x})$. If $r(\boldsymbol{y}) \leq r(\boldsymbol{x})$, then from Lemma 30, $b_{t,n}(\boldsymbol{y}) \leq b_{t,n}(\boldsymbol{x})$. Equivalently, we will show that $wt(\Psi(\boldsymbol{y})) \leq wt(\Psi(\boldsymbol{x}))$. Since $\boldsymbol{y} \in \mathcal{B}_{t,G}(\boldsymbol{x})$ we can write $\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{e_x}$ where $\boldsymbol{e_x}$ is a t-grain-error. Let $\boldsymbol{x}' = \Psi(\boldsymbol{x}), \boldsymbol{e}' = \Psi(\boldsymbol{e_x}), \boldsymbol{y}' = \Psi(\boldsymbol{y})$. By the linearity of the map Ψ, we can write $\boldsymbol{y}' = \boldsymbol{x}' + \boldsymbol{e}'$ and so $wt(\boldsymbol{y}') = wt(\boldsymbol{x}') + wt(\boldsymbol{e}') - 2|supp(\boldsymbol{x}') \cap supp(\boldsymbol{e}')|$. In the following, we show $wt(\boldsymbol{y}') \leq wt(\boldsymbol{x}')$ by proving $|supp(\boldsymbol{x}') \cap supp(\boldsymbol{e}')| \geq \frac{wt(\boldsymbol{e}')}{2}$. The proof will follow by induction on the number of runs of 1s in $\boldsymbol{e_x}$.

We first prove that for any t-grain-error $\boldsymbol{e_x}$ of length n, if $\boldsymbol{e_x}$ has a single run of 1s, then $r(\boldsymbol{y}) \leq r(\boldsymbol{x})$. Suppose then that $\boldsymbol{e_x} = (e_1, \ldots, e_n)$ is a t-grain-error and that $\boldsymbol{e_x}$ contains a single run of 1s. Then $1 \leq wt(\boldsymbol{e}') \leq 2$ since $e_1 = 0$. Suppose further that $\boldsymbol{e}' = (e'_1, \ldots, e'_{n-1})$ has its first 1 at position i where $1 \leq i \leq n-1$. Since i is the location of the first 1 in $\boldsymbol{e}'$, then

$e_i \neq e_{i+1}$ and so $e_i = 0, e_{i+1} = 1$ (since $e_1 = 0$). However, if $e_{i+1} = 1$, then $x_i \neq x_{i+1}$ and so both $x'_i = e'_i = 1$. Since $wt(\boldsymbol{e}') \leq 2$, we have just shown that $|supp(\boldsymbol{x}') \cap supp(\boldsymbol{e}')| \geq 1$, and so the base case is complete.

We now assume that for any length-n $\boldsymbol{e_x}$, if $\boldsymbol{e_x}$ has k runs of 1s, then $r(\boldsymbol{y}) \leq r(\boldsymbol{x})$ where $1 \leq k \leq \lfloor \frac{n}{2} \rfloor$. Consider the case where $\boldsymbol{e_x}$ has $k+1$ runs of 1s. Suppose the k-th run of 1s in $\boldsymbol{e_x}$ has its final 1 in position j where $2 \leq j \leq n-2$. Thus, $e_{j+1} = 0$. For shorthand denote $\boldsymbol{e}_1 = (e_1, \ldots, e_{j+1})$, $\boldsymbol{e}_2 = (e_{j+1}, \ldots, e_n)$, $\boldsymbol{x}_1 = (x_1, \ldots, x_{j+1})$, $\boldsymbol{x}_2 = (x_{j+1}, \ldots, x_n)$, $\boldsymbol{e}'_1 = \Psi(\boldsymbol{e}_1)$, $\boldsymbol{e}'_2 = \Psi(\boldsymbol{e}_2)$, $\boldsymbol{x}'_1 = \Psi(\boldsymbol{x}_1)$, and $\boldsymbol{x}'_2 = \Psi(\boldsymbol{x}_2)$. Notice that the vectors $\boldsymbol{e}'$ and $\boldsymbol{x}'$ can be written as the concatenation of two vectors where $\boldsymbol{e}' = (\boldsymbol{e}'_1, \boldsymbol{e}'_2)$ and $\boldsymbol{x}' = (\boldsymbol{x}'_1, \boldsymbol{x}'_2)$ where $\boldsymbol{e}_1$ is a t-grain-error for $\boldsymbol{x}_1$ with k runs of 1s and $\boldsymbol{e}_2$ is a t-grain-error for $\boldsymbol{x}_2$ with a single run of 1s. By the inductive assumption, $|supp(\boldsymbol{x}'_1) \cap supp(\boldsymbol{e}'_1)| \geq \frac{wt(\boldsymbol{e}'_1)}{2}$ and $|supp(\boldsymbol{x}'_2) \cap supp(\boldsymbol{e}'_2)| \geq \frac{wt(\boldsymbol{e}'_2)}{2}$. Combining these two statements gives the desired result that $|supp(\boldsymbol{x}') \cap supp(\boldsymbol{e}')| \geq \frac{wt(\boldsymbol{e}')}{2}$ and so the proof is complete.

□

The following lemma follows from the structure of grain-errors.

Lemma 32. *For any two vectors $\boldsymbol{x}, \boldsymbol{u} \in GF(2)^n$, suppose that for some $1 \leq i \leq n-1$,*

1. $(x_i, x_{i+1}) = (0,0), (u_i, u_{i+1}) = (1,1)$ *or*

2. $(x_i, x_{i+1}) = (1,1), (u_i, u_{i+1}) = (0,0)$.

Then, $\mathcal{B}_{t,G}(\boldsymbol{x}) \cap \mathcal{B}_{t,G}(\boldsymbol{u}) = \emptyset$.

Proof. Let $\boldsymbol{y}_1 = \boldsymbol{x} + \boldsymbol{e_x}$ and $\boldsymbol{y}_2 = \boldsymbol{u} + \boldsymbol{e_u}$. Since $\boldsymbol{x}$ and $\boldsymbol{u}$ differ at position $i+1$ then in order for $\boldsymbol{y}_1 = \boldsymbol{y}_2$, an error must occur at position $i+1$ in either $\boldsymbol{x}$ or $\boldsymbol{u}$ but not both. However, a grain-error can never change the information at position $i+1$ in either $\boldsymbol{x}$ or $\boldsymbol{u}$ since both $\boldsymbol{x}$ and $\boldsymbol{u}$ store the same information in positions i and $i+1$ by the conditions in the statement of the lemma. □

We now prove the final lemma for this subsection.

Lemma 33. *Suppose $\mathcal{C}$ is a t-grain-error-correcting code of length n with the maximum possible cardinality. Then, $|\mathcal{C}|$ is an even number.*

Proof. Let $\mathcal{C}$ be a t-grain-error-correcting code of length n with the maximum possible cardinality as stated in the lemma. Suppose $\mathcal{G}_G = (V, E)$ is a simple graph where $V = GF(2)^n$. The set E is defined so that a vertex $v_1 \in V$ is adjacent to another vertex $\boldsymbol{v}_2 \in V$ if either $\boldsymbol{v}_1 \in \mathcal{B}_{t,G}(\boldsymbol{v}_2)$ or $\boldsymbol{v}_2 \in \mathcal{B}_{t,G}(\boldsymbol{v}_1)$. Let $V_0 = \{\boldsymbol{v} = (v_1, \ldots, v_n) \in GF(2)^n : v_1 = 0\}$ and $V_1 = \{\boldsymbol{v} = (v_1, \ldots, v_n) \in GF(2)^n : v_1 = 1\}$. Notice first that there are no edges between any of the vertices in V_0 and V_1. Consider the subgraph $\mathcal{G}_0 = (V_0, E_0) \subset \mathcal{G}_G$ where E_0 consists of all the edges from E between the vertices in V_0. Let $\mathcal{G}_1 = (V_1, E_1) \subset \mathcal{G}_G$ where E_1 consists of all the edges from E between vertices in V_1. From Claim 17, $\mathcal{G}_0$ and $\mathcal{G}_1$ are identical so that the maximum number of codewords in $\mathcal{C}$ from $\mathcal{G}_0$ is equal to the maximum number of codewords in $\mathcal{C}$ from $\mathcal{G}_1$ and the statement in the lemma holds. □

The next claim will be used later in Section 5.3 for constructing grain codes.

Claim 19. *Suppose $\mathcal{C}_M$ is a t-mineral-error-correcting code of length n. Let $\mathcal{C}$ be the code that is the result of prepending an arbitrary bit to the beginning of every codeword in $\mathcal{C}_M$. Then, $\mathcal{C}$ is a length-$(n+1)$ t-grain-error-correcting code of size $2|\mathcal{C}_M|$.*

5.1.2 Tools for computing upper bounds

In this subsection, we briefly review some of the tools used in Section 5.2 for computing a non-asymptotic upper bound on the cardinality of grain-error-correcting codes. We begin by revisiting some of the notation and results from [KK12].

Definition 16. *A* **hypergraph** *$\mathcal{H}$ is a pair $(\mathcal{X}, \mathcal{E})$, where $\mathcal{X}$ is a finite set and $\mathcal{E}$ is a collection of nonempty subsets of $\mathcal{X}$ such that $\cup_{E \in \mathcal{E}} E = \mathcal{X}$. The elements of $\mathcal{E}$ are called* **hyperedges**.

Definition 17. *A* **matching** *of a hypergraph $\mathcal{H} = (\mathcal{X}, \mathcal{E})$ is a collection of pairwise disjoint hyperedges $E_1, \ldots, E_j \in \mathcal{E}$. The* **matching number** *of $\mathcal{H}$, denoted $\nu(\mathcal{H})$, is the largest j for which such a matching exists.*

As will be described shortly, the following can be interpreted as the dual of the matching of a hypergraph.

Definition 18. *A* **transversal** *of a hypergraph $\mathcal{H} = (\mathcal{X}, \mathcal{E})$ is a subset $T \subseteq \mathcal{X}$ that intersects every hyperedge in $\mathcal{E}$. The* **transversal number** *of $\mathcal{H}$, denoted by $\tau(\mathcal{H})$, is the smallest size of a transversal.*

Let $\mathcal{H}$ be a hypergraph with vertices $x_1, \ldots, x_n$ and hyperedges $E_1, \ldots, E_m$. The relationships contained within $\mathcal{H}$ can be interpreted through a matrix $A \in \{0,1\}^{n \times m}$, where

$$A(i,j) = \begin{cases} 1 & \text{if } x_i \in E_j, \\ 0 & \text{otherwise,} \end{cases}$$

for $1 \leq i \leq n, 1 \leq j \leq m$. Cast in this light, the matching number and the transversal number can be derived using linear optimization techniques.

Lemma 34. *(cf. [KK12]) The matching number and the transversal number are the solutions of the integer linear programs:*

$$\nu(\mathcal{H}) = \max\{\mathbf{1^T} \boldsymbol{z} | A\boldsymbol{z} \leq \mathbf{1}, z_j \in \{0,1\}, 1 \leq j \leq m\}, \text{ and} \tag{5.1}$$

$$\tau(\mathcal{H}) = \min\{\mathbf{1^T} \boldsymbol{u} | A^T \boldsymbol{u} \geq \mathbf{1}, u_i \in \{0,1\}, 1 \leq i \leq n\}, \tag{5.2}$$

where $\mathbf{1}$ *denotes a column vector of all* 1*s of the appropriate dimension.*

Relaxing the condition that the solutions to the programming problem are comprised of

0s and 1s, we have the following problems:

$$\nu^*(\mathcal{H}) = \max\{\mathbf{1^T}\boldsymbol{z} | A\boldsymbol{z} \leq \mathbf{1}, \boldsymbol{z} \geq 0\}, \text{ and} \tag{5.3}$$

$$\tau^*(\mathcal{H}) = \min\{\mathbf{1^T}\boldsymbol{u} | A^T\boldsymbol{u} \geq \mathbf{1}, \boldsymbol{u} \geq 0\}. \tag{5.4}$$

Clearly $\nu(\mathcal{H}) \leq \nu^*(\mathcal{H})$ and $\tau(\mathcal{H}) \geq \tau^*(\mathcal{H})$. Since (5.3) and (5.4) are linear programs, they satisfy strong duality [BV04] and $\nu^*(\mathcal{H}) = \tau^*(\mathcal{H})$. Thus, combining these inequalities leads us to $\nu(\mathcal{H}) \leq \tau^*(\mathcal{H})$ [KK12].

5.1.3 Graph notation

In this subsection, we describe graph notation from [WES01] that will be used in Section 5.3.2 and Section 5.5. Let $\mathcal{G} = (V, E)$ be a simple graph; that is, it has undirected edges with no parallel edges and no self-loops. A vertex $\boldsymbol{v}_1 \in V$ is adjacent to another vertex $\boldsymbol{v}_2 \in V$ if there exists an edge between them. The degree of a vertex is the number of its adjacent vertices and the maximum degree of a vertex in $\mathcal{G}$ is denoted $\Delta(\mathcal{G})$.

A ***k-coloring*** is a mapping $\Phi : V \to \{0, 1, \ldots, k-1\}$ of numbers to each vertex such that the same number is never assigned to adjacent vertices. The ***chromatic number*** of a graph, denoted by $\chi(\mathcal{G})$, is the smallest k for which a k-coloring exists. A ***clique*** is a set of vertices in $\mathcal{G}$ that are all adjacent. The size of the largest clique in a graph $\mathcal{G}$ is denoted $\varsigma(\mathcal{G})$. It is known that for a graph $\mathcal{G}$, $\chi(\mathcal{G})$ is such that $\varsigma(\mathcal{G}) \leq \chi(\mathcal{G}) \leq \Delta(\mathcal{G}) + 1$ [WES01]. Each collection of vertices that share the same number (under some fixed k-coloring) is referred to as a ***color class***.

5.1.4 Distance metrics and group codes

In this subsection, we introduce some distance metrics that are used in Section 5.3 to construct grain-error-correcting codes. In addition, we define group codes that will serve as the foundation of the single grain-error-correcting codes introduced in Section 5.3.1.

Definition 19. *Suppose* $\boldsymbol{x}, \boldsymbol{y} \in GF(2)^n$. *Their* **Hamming distance** *is denoted* $d_H(\boldsymbol{x}, \boldsymbol{y}) = |\{i : x_i \neq y_i\}|$.

Definition 20. *Suppose* $\boldsymbol{x} = (x_1, \ldots, x_n), \boldsymbol{y} = (y_1, \ldots, y_n) \in GF(2)^n$. *For* $1 \leq i \leq n$, $N(\boldsymbol{x}, \boldsymbol{y}) = |\{i : x_i > y_i\}|$.

Definition 21. *(cf. [CR79]) Suppose* $\boldsymbol{x}, \boldsymbol{y}$ *are two vectors in* $GF(2)^n$. *Their* **asymmetric distance** *is denoted* $d_A(\boldsymbol{x}, \boldsymbol{y}) = \max\{N(\boldsymbol{x}, \boldsymbol{y}), N(\boldsymbol{y}, \boldsymbol{x})\}$.

We say that a code $\mathcal{C}$ has ***minimum Hamming distance*** $d_H(\mathcal{C})$ if $d_H(\mathcal{C})$ is the smallest Hamming distance between any two distinct codewords in $\mathcal{C}$. Similarly, we say that a code $\mathcal{C}$ has ***minimum asymmetric distance*** $d_A(\mathcal{C})$ if $d_A(\mathcal{C})$ is the smallest asymmetric distance between any two distinct codewords in $\mathcal{C}$.

Suppose $\mathcal{A}$ is an additive Abelian group of order $n+1$ and suppose $(\tilde{g}_1, \ldots, \tilde{g}_n)$ is a sequence consisting of the distinct non-zero elements of $\mathcal{A}$. For every $a \in \mathcal{A}$, we define a ***group code*** $\tilde{\mathcal{C}}_a^{\mathcal{A}}$ to be

$$\tilde{\mathcal{C}}_a^{\mathcal{A}} = \{\boldsymbol{x} \in GF(2)^n | \sum_{k=1}^{n} x_k \tilde{g}_k = a\}.$$

Without loss of generality, we assume the Abelian groups we deal with in this chapter are additive, and that the group operation is denoted as addition. Such a construction was shown in [CR79] to have $d_H(\tilde{\mathcal{C}}_a^{\mathcal{A}}) \geq d_A(\tilde{\mathcal{C}}_a^{\mathcal{A}}) \geq 2$. We include the following example for clarity.

Example 14. *Let* $\mathcal{A}$ *be the additive Abelian group* $\mathbb{Z}_3$ *so that* $(\tilde{g}_1, \tilde{g}_2) = (1, 2)$. *Then, the group* $\mathcal{A}$ *partitions the space* $GF(2)^2$ *into* 3 *group codes.*

$$\tilde{\mathcal{C}}_0^{\mathbb{Z}_3} = \{(0,0), (1,1)\},$$
$$\tilde{\mathcal{C}}_1^{\mathbb{Z}_3} = \{(1,0)\},$$
$$\tilde{\mathcal{C}}_2^{\mathbb{Z}_3} = \{(0,1)\}.$$

An ***elementary Abelian group*** is a finite Abelian group where every non-identity element in the group has order p, where p is a prime. For shorthand, the elementary Abelian

group of size p^r (for a prime p and a positive integer r) is referred to as an ***elementary Abelian p-group*** [ROT94].

5.1.5 Discrete Fourier analysis

In this subsection, we briefly review some of the tools that will be used in Section 5.4 to derive lower bounds on the cardinalities of code constructions. The notation adopted is similar to the notation used in [MR79].

Let p be a prime number and suppose ζ_p denotes the complex primitive p-th root of unity and suppose r is some positive integer. Let $\mathcal{A}$ refer to the additive Abelian group $(\mathbb{Z}_p)^r = \underbrace{\mathbb{Z}_p \times \cdots \times \mathbb{Z}_p}_{r \text{ times}}$. The operator $< \boldsymbol{g}, \boldsymbol{h} >$ takes two elements $\boldsymbol{g} = (g_1, \ldots, g_r), \boldsymbol{h} = (h_1, \ldots, h_r) \in \mathcal{A}$ and maps them into a complex number as follows

$$< \boldsymbol{g}, \boldsymbol{h} >= \prod_{i=1}^{r} (\zeta_p)^{g_i h_i} = (\zeta_p)^{\sum_{i=1}^{r} g_i h_i} = (\zeta_p)^{\boldsymbol{g}^T \cdot \boldsymbol{h}}.$$

Let $f(\boldsymbol{g})$ be any function that maps elements of $\mathcal{A}$ into the complex plane. The ***Fourier transform*** $\hat{f}$ of f is defined as

$$\hat{f}(\boldsymbol{h}) = \sum_{\boldsymbol{g} \in \mathcal{A}} < \boldsymbol{h}, -\boldsymbol{g} > f(\boldsymbol{g})$$

and the ***inverse Fourier transform*** is defined as

$$f(\boldsymbol{g}) = \frac{1}{p^r} \sum_{\boldsymbol{h} \in \mathcal{A}} < \boldsymbol{h}, \boldsymbol{g} > \hat{f}(\boldsymbol{h}).$$

5.2 Upper Bounds on Grain-Error Codes

In this section, we use linear programming methods to produce a closed-form upper bound on the cardinality of a t-grain-error-correcting code. The approach is analogous to that found in [KK12] where upper bounds were computed for the deletion channel and in [KZ13] where

upper bounds were derived for the non-overlapping grain-error model. Recall, our objective is to compute upper bounds for the overlapping grain-error model.

The approach is the following. First, the vector space from which codewords are chosen, is projected onto a hypergraph. Then, an approximate solution to a matching problem is derived. Recall that the maximum size of a t-grain-error-correcting-code of length n will be referred to as $M(n,t)$.

Let $\mathcal{H}_{t,n}$ denote the hypergraph for a t-grain-error-correcting code. More formally, let

$$\mathcal{H}_{t,n} = (GF(2)^n, \{\mathcal{B}_{t,G}(\boldsymbol{x}) | \boldsymbol{x} \in GF(2)^n\}).$$

In this graph, the vertices represent candidate codewords and the hyperedges represent vectors that result when t or fewer grain-errors occur in any of the candidate codewords.

As in [KK12], $\nu^*(\mathcal{H}_{t,n})$ is an upper bound on $M(n,t)$ and will be derived by considering the dual problem defined in (4). The problem is to find a function $w : GF(2)^n \to \mathbb{R}^+$ such that

$$\tau^*(\mathcal{H}_{t,n}) = \min_w \{ \sum_{\boldsymbol{y} \in GF(2)^n} w(\boldsymbol{y}) \}$$

subject to

$$\sum_{\boldsymbol{y} \in \mathcal{B}_{t,G}(\boldsymbol{x})} w(\boldsymbol{y}) \geq 1, \forall \boldsymbol{x} \in GF(2)^n \tag{5.5}$$

and $w(\boldsymbol{x}) \geq 0, \forall \boldsymbol{x} \in GF(2)^n$.

We are now ready to state the main result of the section.

Theorem 10. *For positive integers n, t where $t < n$,*

$$M(n,t) \leq 2\sum_{k=0}^{n-1} \begin{pmatrix} n-1 \\ k \end{pmatrix} \frac{1}{\sum_{j=0}^{\min\{t,k\}} \begin{pmatrix} k \\ j \end{pmatrix}}.$$

Proof. In order to prove the result, we must assign values for $w(\boldsymbol{y})$ such that the constraint in (5.5) is satisfied. Let $w(\boldsymbol{y}) = \frac{1}{b_{t,n}(\boldsymbol{y})}$ where $b_{t,n}(\boldsymbol{y})$ is computed as in Lemma 30. Note that

$$\sum_{\boldsymbol{y} \in \mathcal{B}_{t,G}(\boldsymbol{x})} w(\boldsymbol{y}) = \sum_{\boldsymbol{y} \in \mathcal{B}_{t,G}(\boldsymbol{x})} \frac{1}{b_{t,n}(\boldsymbol{y})}.$$

From Lemma 31, for any $\boldsymbol{y} \in \mathcal{B}_{t,G}(\boldsymbol{x})$, $b_{t,n}(\boldsymbol{y}) \leq b_{t,n}(\boldsymbol{x})$, so we have

$$\sum_{\boldsymbol{y} \in \mathcal{B}_{t,G}(\boldsymbol{x})} \frac{1}{b_{t,n}(\boldsymbol{y})} \geq \sum_{\boldsymbol{y} \in \mathcal{B}_{t,G}(\boldsymbol{x})} \frac{1}{b_{t,n}(\boldsymbol{x})} = b_{t,n}(\boldsymbol{x}) \frac{1}{b_{t,n}(\boldsymbol{x})} = 1.$$

The theorem statement now follows from the bound on $\sum_{\boldsymbol{y} \in GF(2)^n} w(\boldsymbol{y})$: Since the number of length-n vectors with k runs is $2\begin{pmatrix} n-1 \\ k-1 \end{pmatrix}$ and $b_{t,n} = \sum_{j=0}^{\min\{t,k-1\}} \begin{pmatrix} k-1 \\ j \end{pmatrix}$ from Lemma 1, we have

$$M(n,t) \leq 2\sum_{k=1}^{n} \begin{pmatrix} n-1 \\ k-1 \end{pmatrix} \frac{1}{\sum_{j=0}^{\min\{t,k-1\}} \begin{pmatrix} k-1 \\ j \end{pmatrix}},$$

which, after reindexing the parameter k, is the statement in the theorem. □

Theorem 10 gives an explicit upper bound on $M(n,t)$ for all n and t. However, providing an explicit expression (without summations) is still not easy to derive. In the following, we present non-asymptotic bounds for $t = 1, 2, 3$. The bounds will then be compared against the existing bounds in [SR11] for $t = 1, 2, 3$. Note that the overlapping and non-overlapping

grain-error models coincide for the case where $t = 1$. The following corollary was also derived in [KZ13] in the context of the non-overlapping grain-error model. It is the result of combining Theorem 10 for the case where $t = 1$ with Lemma 33. Recall, $M(n, t)$ refers to the maximum size of a t-grain-error-correcting code.

Corollary 12. *For $n \geq 1$, $M(n, 1) \leq 2\lfloor \frac{2^{n+1}-2}{2n} \rfloor$.*

In general, it is difficult to compare our bounds to those in [SR11] since the bounds in [SR11] require finding a parameter ρ where ρ is the largest integer satisfying

$$\sum_{k=1}^{\rho} \begin{pmatrix} n-1 \\ k-1 \end{pmatrix} \sum_{j=0}^{\min(t,k)} \begin{pmatrix} k \\ j \end{pmatrix} \leq 2^{n-1}.$$

The bounds for $t = 1$ and small n were explicitly derived using the formula in [SR11] and for all values of $n \leq 20$ the bound in Corollary 12 was tighter (as can be seen in Table 5.2). The bounds in this section have the advantage of being explicit.

For the case of $t = 2$, we make use of the following claims which can be proven using induction. We begin with the following claims.

Claim 20. *For $n \geq 2$,*

$$\sum_{k=2}^{n} \frac{1}{k+1} \begin{pmatrix} n \\ k \end{pmatrix} = \frac{1}{n+1} \left(2^{n+1} - 2 - \frac{3n}{2} - \frac{n^2}{2} \right).$$

Proof. This identity follows from the following derivations:

$$\begin{aligned} &\sum_{k=2}^{n} \frac{1}{k+1} \begin{pmatrix} n \\ k \end{pmatrix} = \sum_{k=0}^{n} \frac{1}{k+1} \begin{pmatrix} n \\ k \end{pmatrix} - 1 - n/2 \\ =&\sum_{k=0}^{n} \frac{1}{n+1} \begin{pmatrix} n+1 \\ k+1 \end{pmatrix} - 1 - n/2 = \frac{2^{n+1}-1}{n+1} - 1 - n/2 \\ =&\frac{1}{n+1}(2^{n+1} - 2 - \frac{3n}{2} - \frac{n^2}{2}). \end{aligned}$$

□

The next lemma will be useful to obtain the expression for Claim 21.

Lemma 35. *For integers n, c where $0 \leq c < n$,*

$$\sum_{k=c+1}^{n} \frac{1}{k-c} \binom{n}{k} = \sum_{j=1}^{n-c} \frac{\binom{n-j}{c}}{j} \cdot \left(2^j - 1\right).$$

Proof. Using a change of variable and since $\binom{n}{k+c} = \sum_{j=k}^{n-c} \binom{j-1}{k-1} \binom{n-j}{c}$, we have

$$\begin{aligned}\sum_{k=c+1}^{n} \frac{1}{k-c} \binom{n}{k} &= \sum_{k=1}^{n-c} \frac{1}{k} \cdot \sum_{j=k}^{n-c} \binom{j-1}{k-1} \binom{n-j}{c} \\ &= \sum_{k=1}^{n-c} \sum_{j=k}^{n-c} \frac{1}{j} \binom{j}{k} \binom{n-j}{c}.\end{aligned}$$

Reversing the order of the summations gives

$$\begin{aligned}\sum_{k=1}^{n-c} \sum_{j=k}^{n-c} \frac{1}{j} \binom{j}{k} \binom{n-j}{c} &= \sum_{j=1}^{n-c} \frac{\binom{n-j}{c}}{j} \sum_{k=1}^{j} \binom{j}{k} \\ &= \sum_{j=1}^{n-c} \frac{\binom{n-j}{c}}{j} (2^j - 1),\end{aligned}$$

as desired. □

As a consequence of the previous lemma we have the following corollary.

Corollary 13. *For $n \geq 2$, $\sum_{k=1}^{n} \frac{1}{k} \begin{pmatrix} n \\ k \end{pmatrix} = \sum_{j=1}^{n} \frac{1}{j}(2^j - 1)$.*

We require one more lemma before proceeding to the proof of Claim 21.

Lemma 36. *For $n \geq 17$,*

$$\sum_{k=1}^{n} \frac{2^k - 1}{k} \leq \frac{2^{n+1}}{n - 1 - \frac{2}{n-5} + \frac{1}{n^2}}.$$

Proof. For $n = 17$, the value on the left hand side is equal 16552.47, while the value of the right hand side is equal 16552.85. Now, assume the inequality holds for some $n \geq 17$, and we will show its validity for $n + 1$. Hence, we need to show that

$$\sum_{k=1}^{n+1} \frac{2^k - 1}{k} \leq \frac{2^{n+2}}{n - \frac{2}{n-4} + \frac{1}{(n+1)^2}}.$$

According to the induction assumption, it is enough to show that

$$\frac{2^{n+1}}{n - 1 - \frac{2}{n-5} + \frac{1}{n^2}} + \frac{2^{n+1} - 1}{n+1} \leq \frac{2^{n+2}}{n - \frac{2}{n-4} + \frac{1}{(n+1)^2}},$$

or

$$\frac{1}{n - 1 - \frac{2}{n-5} + \frac{1}{n^2}} + \frac{1}{n+1} \leq \frac{2}{n - \frac{2}{n-4} + \frac{1}{(n+1)^2}},$$

which holds for $n \geq 17$. □

Claim 21 now follows from Corollary 13 and Lemma 36.

Claim 21. *For $n \geq 17$,*

$$\sum_{k=1}^{n} \frac{1}{k} \begin{pmatrix} n \\ k \end{pmatrix} \leq \frac{2^{n+1}}{n - 1 - \frac{2}{n-5} + \frac{1}{n^2}}.$$

We now derive the bound for $M(n, 2)$, the maximum size of a 2-grain-error-correcting code, which is non-asymptotic and explicit.

Lemma 37. *For $n \geq 14$, $M(n, 2) \leq 2\left\lfloor \frac{2^{n+2}(2 + \frac{2}{n-6})}{2n(n-3)} \right\rfloor$.*

Proof. From Theorem 10 we have

$$M(n,2) = 2\sum_{k=0}^{n-1} \binom{n-1}{k} \frac{1}{\sum_{j=0}^{\min\{2,k\}} \binom{k}{j}}$$
$$= 2 + n - 1 + 2\sum_{k=2}^{n-1} \binom{n-1}{k} \frac{1}{1+k+\binom{k}{2}}$$
$$\leq n + 1 + 4\sum_{k=2}^{n-1} \binom{n-1}{k} \left(\frac{1}{k} - \frac{1}{k+1}\right)$$
$$= n + 1 + 4\sum_{k=2}^{n-1} \binom{n-1}{k} \frac{1}{k} - 4\sum_{k=2}^{n-1} \binom{n-1}{k} \frac{1}{k+1}.$$

From Claims 20 and 21 we have

$$\begin{aligned} M(n,2) &\leq n + 1 + 4\left(\frac{2^n}{n-2-\frac{2}{n-6}} - n + 1\right) \\ &\quad - \frac{4}{n}\left(2^n - 2 - \frac{3(n-1)}{2} - \frac{(n-1)^2}{2}\right) \\ &= \frac{2^{n+2}}{n-2-\frac{2}{n-6}} - \frac{2^{n+2}}{n} - n + 7 + \frac{4}{n} \\ &\leq \frac{2^{n+2}(2+\frac{2}{n-6})}{n(n-3)}. \end{aligned}$$

From Lemma 33 $M(n,2)$ must be an even integer and so $M(n,2) \leq 2\left\lfloor \frac{2^{n+2}(2+\frac{2}{n-6})}{2n(n-3)} \right\rfloor$. □

We now proceed to derive similar results for $M(n,3)$. We first note the following corollary which follows from Lemma 35.

Corollary 14. *For $n \geq 2$,*

$$\sum_{k=2}^{n} \frac{1}{k-1} \binom{n}{k} = n \sum_{k=1}^{n-1} \frac{2^k - 1}{k} - 2^n + n + 1.$$

Proof. From Lemma 35, we have

$$\begin{aligned}\sum_{k=2}^{n} \frac{1}{k-1} \left(\begin{array}{c} n \\ k \end{array} \right) &= \sum_{j=1}^{n-1} \frac{n-j}{j} \cdot (2^j - 1) \\ &= n \sum_{j=1}^{n-1} \frac{2^j - 1}{j} - \sum_{j=1}^{n-1} (2^j - 1) \\ &= n \sum_{j=1}^{n-1} \frac{2^j - 1}{j} - ((2^n - 1) - 1) + n - 1,\end{aligned}$$

which simplifies to the expression in the corollary. □

Lemma 38. *For $n \geq 24$,*

$$M(n,3) \leq 2 \left\lfloor 2^n \left(\frac{\frac{36n}{n-7} + 18 - \frac{3n(n-1)}{(n-2)^2} + \frac{12}{n-7}}{2n(n-1)(n-3-\frac{2}{n-7}+\frac{1}{(n-2)^2}} \right) \right\rfloor.$$

Proof. From Theorem 10 we have

$$\begin{aligned}M(n,3) &\leq 2 \sum_{k=0}^{n-1} \left(\begin{array}{c} n-1 \\ k \end{array} \right) \frac{1}{\sum_{j=0}^{\min\{3,k\}} \left(\begin{array}{c} k \\ j \end{array} \right)} \\ &= 2 + n - 1 + \frac{1}{2} \cdot \binom{n-1}{2} + 2 \sum_{k=3}^{n-1} \left(\begin{array}{c} n-1 \\ k \end{array} \right) \frac{1}{1 + k + \binom{k}{2} + \binom{k}{3}} \\ &\leq \frac{n^2+n+6}{4} + 2 \sum_{k=3}^{n-1} \left(\begin{array}{c} n-1 \\ k \end{array} \right) \frac{1}{\binom{k}{2} + \binom{k}{3}} \\ &= \frac{n^2+n+6}{4} + 12 \sum_{k=3}^{n-1} \left(\begin{array}{c} n-1 \\ k \end{array} \right) \left(\frac{1/2}{k-1} - \frac{1}{k} + \frac{1/2}{k+1} \right) \\ &= \frac{n^2+n+6}{4} + 6 \sum_{k=3}^{n-1} \left(\begin{array}{c} n-1 \\ k \end{array} \right) \frac{1}{k-1} \\ &\quad - 12 \sum_{k=3}^{n-1} \left(\begin{array}{c} n-1 \\ k \end{array} \right) \frac{1}{k} + 6 \sum_{k=3}^{n-1} \left(\begin{array}{c} n-1 \\ k \end{array} \right) \frac{1}{k+1}.\end{aligned}$$

According to Corollary 14,

$$\sum_{k=3}^{n-1}\binom{n-1}{k}\frac{1}{k-1}=(n-1)\sum_{k=1}^{n-2}\frac{2^k-1}{k}-2^{n-1}-\frac{n^2-5n+2}{2}.$$

By Corollary 13,

$$\sum_{k=3}^{n-1}\binom{n-1}{k}\frac{1}{k}=\sum_{k=1}^{n-1}\frac{2^k-1}{k}-\frac{n^2+n-2}{4},$$

and by Claim 20,

$$\sum_{k=3}^{n-1}\binom{n-1}{k}\frac{1}{k+1}=\frac{1}{n}\left(2^n-2-\frac{3(n-1)}{2}-\frac{(n-1)^2}{2}\right)-\frac{n^2-3n+2}{6}.$$

All together we get that

$$\begin{aligned}
M(n,3)&\le\frac{n^2+n+6}{4}\\
&+6\left((n-1)\sum_{k=1}^{n-2}\frac{2^k-1}{k}-2^{n-1}-\frac{n^2-5n+2}{2}\right)\\
&-12\left(\sum_{k=1}^{n-1}\frac{2^k-1}{k}-\frac{n^2+n-2}{4}\right)\\
&+6\left(\frac{1}{n}\left(2^n-2-\frac{3(n-1)}{2}-\frac{(n-1)^2}{2}\right)-\frac{n^2-3n+2}{6}\right)\\
&=-\frac{3n^2}{4}+\frac{73n}{4}-\frac{31}{2}-\frac{6}{n}+6(n-1)\sum_{k=1}^{n-2}\frac{2^k-1}{k}-3\cdot 2^n\\
&-12\sum_{k=1}^{n-1}\frac{2^k-1}{k}+\frac{6\cdot 2^n}{n}\\
&=-\frac{3n^2}{4}+\frac{73n}{4}-\frac{31}{2}-\frac{6}{n}+6(n-3)\sum_{k=1}^{n-2}\frac{2^k-1}{k}-3\cdot 2^n\\
&-12\cdot\frac{2^{n-1}-1}{n-1}+\frac{6\cdot 2^n}{n}\\
&\le 6(n-3)\sum_{k=1}^{n-2}\frac{2^k-1}{k}-3\cdot 2^n-\frac{6\cdot 2^n}{n-1}+\frac{6\cdot 2^n}{n}\\
&=6(n-3)\sum_{k=1}^{n-2}\frac{2^k-1}{k}-3\cdot 2^n-\frac{6\cdot 2^n}{n(n-1)}
\end{aligned}$$

where the inequality holds for $n \geq 24$. Finally, according to Lemma 36 we get

$$
\begin{aligned}
M(n,3) &\leq 6(n-3)\frac{2^{n-1}}{n-3-\frac{2}{n-7}+\frac{1}{(n-2)^2}} - 3\cdot 2^n - \frac{6\cdot 2^n}{n(n-1)} \\
&= 2^n\left(\frac{3n-9}{n-3-\frac{2}{n-7}+\frac{1}{(n-2)^2}} - 3 - \frac{6}{n(n-1)}\right) \\
&= 2^n\left(\frac{\frac{6}{n-7}-\frac{3}{(n-2)^2}}{n-3-\frac{2}{n-7}+\frac{1}{(n-2)^2}} - \frac{6}{n(n-1)}\right) \\
&= 2^n\left(\frac{\frac{6(n(n-1))}{n-7} - 6n - \frac{3n(n-1)}{(n-2)^2} + 18 + \frac{12}{n-7} - \frac{6}{(n-2)^2}}{n(n-1)(n-3-\frac{2}{n-7}+\frac{1}{(n-2)^2})}\right) \\
&\leq 2^n\left(\frac{\frac{6(n(n-1))}{n-7} - 6n + 18 - \frac{3n(n-1)}{(n-2)^2} + \frac{12}{n-7}}{n(n-1)(n-3-\frac{2}{n-7}+\frac{1}{(n-2)^2})}\right) \\
&= 2^n\left(\frac{\frac{36n}{n-7} + 18 - \frac{3n(n-1)}{(n-2)^2} + \frac{12}{n-7}}{n(n-1)(n-3-\frac{2}{n-7}+\frac{1}{(n-2)^2}}\right).
\end{aligned}
$$

From Lemma 33, $M(n,3)$ must be an even integer and so $M(n,3) \leq 2\left\lfloor 2^n\left(\frac{\frac{36n}{n-7}+18-\frac{3n(n-1)}{(n-2)^2}+\frac{12}{n-7}}{2n(n-1)(n-3-\frac{2}{n-7}+\frac{1}{(n-2)^2}}\right)\right\rfloor$.

We briefly note that Lemma 38 provides a looser asymptotic upper bound on $M(n,t)$ for the case where $t=3$ than the bound provided in [MBK11]. However, Lemma 38 has the advantage of providing a non-asymptotic expression.

In Table 5.1, we illustrate the result for $M(n,t)$ for small n and $t = 1,2,3$ from Theorem 10. Each entry in Table 5.1 consists of a pair of entries delimited by a '/'. The first entry is the previous best value for $M(n,t)$ taken from [SR11] and the second value is the result of using Theorem 10. In general, it is difficult to compare our bounds to those in [SR11] since the bounds in [SR11] require finding a parameter ρ where ρ is the largest integer satisfying $\sum_{k=0}^{\rho-1}\binom{n-1}{k}\sum_{j=0}^{\min(t,k)}\binom{k}{j} \leq 2^{n-1}$. For $t = 1,2,3$ and small n, the values of $M(n,t)$ from [SR11] were computed, and for all values of $n \leq 20$ the bound in Corollary 12 was tighter (as can be seen in Table 5.1). In addition, the bounds in this section have the advantage of being explicit.

□

Table 5.1: Comparison of Existing Upper Bounds from [SR11] and New Upper Bounds from Theorem 10

Length	$M(n,1)$		$M(n,2)$		$M(n,3)$	
3	4	**4**	-	-	-	-
4	6	**6**	4	6	-	-
5	8	12	8	10	-	-
6	16	20	10	14	8	14
7	26	36	16	24	16	22
8	44	62	22	38	18	34
9	150	**112**	110	**62**	32	52
10	278	**204**	190	**102**	178	**80**
11	506	**372**	346	**168**	316	**126**
12	942	**682**	582	**280**	528	**198**
13	1760	**1260**	1010	**476**	870	**312**
14	3256	**2340**	1818	**814**	1498	**496**
15	6148	**4368**	3246	**1406**	2682	**800**
16	11532	**8190**	5646	**2448**	4512	**1300**
17	21654	**15420**	10168	**4302**	7590	**2132**
18	41340	**29126**	18852	**7612**	13300	**3528**
19	77792	**55188**	32962	**13560**	24178	**5892**
20	147788	**104856**	59518	**24306**	40724	**9920**

5.3 Grain-Error Code Constructions

In the previous section, the focus was on upper bounds for grain-error-correcting codes. In this section, we turn to code constructions. We will compare the codes proposed in this section to the upper bounds derived in the previous section.

This section is divided into three subsections. In Section 5.3.1, we consider a group-theoretic construction for single-grain codes. In Section 5.3.2, we generalize the construction from 5.3.1. Using this generalization, Section 5.3.2 identifies better codes that correct single grain-errors for certain code lengths. Section 5.3.3 considers constructions for codes that can correct multiple grain-errors.

5.3.1 Single-grain codes

We begin by proving some sufficient conditions for a code to correct a single grain-error. Then, we provide a group-theoretic code construction that satisfies these conditions. The codes presented in this section provide the largest known cardinalities for all code lengths greater than 16.

Combining Lemma 32 with Definition 14, the following claim can be verified. Recall that d_H and d_A refer to the Hamming distance and the asymmetric distance, respectively.

Claim 22. *A code $\mathcal{C}$ is a single-grain code if for every pair of distinct codewords $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{C}$ one of the following holds:*

1. $d_H(\boldsymbol{x}, \boldsymbol{y}) = 1$ *and* $x_1 \neq y_1$.
2. $d_H(\boldsymbol{x}, \boldsymbol{y}) = 2$ *and for some* $1 < i \leq n-1$,
   (a) $(x_i, x_{i+1}) = (0,0), (y_i, y_{i+1}) = (1,1)$ *or*
   (b) $(x_i, x_{i+1}) = (1,1), (y_i, y_{i+1}) = (0,0)$.
3. $d_H(\boldsymbol{x}, \boldsymbol{y}) \geq 3$.

We are now ready to state our code construction. For any Abelian group referred to in the subsequent discussion, the identity element will be denoted as 0 and will be referred to as the zero element.

Construction E. *Let $\mathcal{A}$ represent an additive Abelian group of size n. Suppose the sequence $\mathcal{S} = (g_1, g_2, \ldots, g_n)$, which contains every element of $\mathcal{A}$ once, is ordered as follows:*

1. $g_1 = 0$,
2. *for any $1 < i \leq n$, the elements g_i and $-g_i$ (if $-g_i \neq g_i$) are adjacent.*

For any element $a \in \mathcal{A}$, let

$$\mathcal{C}_a^{\mathcal{A}} = \{\boldsymbol{x} \in \{0,1\}^n : \sum_{k=1}^{n} x_k g_k = a\}. \tag{5.6}$$

The following example illustrates Construction E.

Example 15. *Let $\mathcal{A}$ denote the additive Abelian group $\mathbb{Z}_3$. Suppose Construction E is used to create a code where $\mathcal{S} = (g_1, g_2, g_3) = (0, 1, 2)$. Then, the group $\mathcal{A}$ partitions the space $GF(2)^3$ into 3 single-grain codes.*

$$\mathcal{C}_0^{\mathbb{Z}_3} = \{(0,0,0), (1,0,0), (0,1,1), (1,1,1)\},$$
$$\mathcal{C}_1^{\mathbb{Z}_3} = \{(0,1,0), (1,1,0)\},$$
$$\mathcal{C}_2^{\mathbb{Z}_3} = \{(0,0,1), (1,0,1)\}.$$

The correctness of Construction E is proven next.

Theorem 11. *A code $\mathcal{C}_a^{\mathcal{A}}$ created with Construction E is a single-grain code.*

Proof. We will show that $\mathcal{C}_a^{\mathcal{A}}$ is a single-grain code by demonstrating that the conditions listed in Claim 22 hold for any pair of distinct codewords $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{C}_a^{\mathcal{A}}$. Let $\tilde{\mathcal{C}}_a^{\mathcal{A}}$ be the group code created by using the same group and element a as in $\mathcal{C}_a^{\mathcal{A}}$ so that $\tilde{\mathcal{C}}_a^{\mathcal{A}}$ has length $n-1$, and $\tilde{\mathcal{C}}_a^{\mathcal{A}}$ is obtained by shortening the codewords of $\mathcal{C}_a^{\mathcal{A}}$ on the first bit (i.e., by removing x_1, which multiplies $g_1 = 0$). Recall from Section 5.1.4 that since $\tilde{\mathcal{C}}_a^{\mathcal{A}}$ is a group code, $d_H(\tilde{\mathcal{C}}_a^{\mathcal{A}}) \geq d_A(\tilde{\mathcal{C}}_a^{\mathcal{A}}) \geq 2$.

Suppose $d_H(\boldsymbol{x}, \boldsymbol{y}) = 1$. Then, since $d_H(\tilde{\mathcal{C}}_a^{\mathcal{A}}) \geq 2$, it follows that if $d_H(\boldsymbol{x}, \boldsymbol{y}) = 1$, then $\boldsymbol{x}$ and $\boldsymbol{y}$ differ only in the first bit and so condition 1) from Claim 22 holds.

Suppose $d_H(\boldsymbol{x}, \boldsymbol{y}) = 2$. Since $d_H(\tilde{\mathcal{C}}_a^{\mathcal{A}}) \geq 2$, $\boldsymbol{x}$ and $\boldsymbol{y}$ do not differ in the first position, and there are two distinct indices i, j $(2 \leq i, j \leq n)$ where $x_i \neq y_i$ and $x_j \neq y_j$. Suppose, without loss of generality, that $N(\boldsymbol{x}, \boldsymbol{y}) = 2$ and so $x_i = x_j = 1$. Therefore, $g_i + g_j = 0$, or $g_j = g_i^{-1}$.

However, by condition 2) in Construction E, we have $|j - i| = 1$ and so condition 2) from Claim 22 holds.

If $d_H(\boldsymbol{x}, \boldsymbol{y})$ is not equal to 1 or 2 then $d_H(\boldsymbol{x}, \boldsymbol{y}) \geq 3$ and so condition 3) of Claim 22 holds. □

The following corollary follows from the proof of Theorem 11 and Claim 17.

Corollary 15. *Let $\mathcal{C}_a^{\mathcal{A}}$ be a single-grain code created according to Construction E. Let $\tilde{\mathcal{C}}_a^{\mathcal{A}}$ be the group code that is the result of shortening the codewords in $\mathcal{C}_a^{\mathcal{A}}$ on the first bit. Then $\tilde{\mathcal{C}}_a^{\mathcal{A}}$ is a single-mineral code.*

The following corollary provides upper and lower bounds on $|\mathcal{C}_a^{\mathcal{A}}|$.

Corollary 16. *Suppose $\mathcal{A}$ is an Abelian group of size n and $a \in \mathcal{A}$. Then, for a code $\mathcal{C}_a^{\mathcal{A}}$ created according to Construction E, $|\mathcal{C}_a^{\mathcal{A}}| \leq |\mathcal{C}_0^{\mathcal{A}}|$. Furthermore,*

$$\frac{2^n}{n} \leq |\mathcal{C}_0^{\mathcal{A}}| \leq \frac{2^n}{n} + \frac{(n-1) \cdot 2^{n/3}}{n}.$$

Equality holds on the left if and only if $|\mathcal{A}|$ is a power of two. Equality holds on the right if and only if $\mathcal{A}$ is an elementary Abelian 3-group.

Proof. Since Construction E concatenates an arbitrary bit with a group code, it follows that if the underlying group code of length $n' = n-1$ has cardinality $|\tilde{\mathcal{C}}_a^{\mathcal{A}}|$, then the code $\mathcal{C}_a^{\mathcal{A}}$ created using the previous construction has $2|\tilde{\mathcal{C}}_a^{\mathcal{A}}|$ codewords. Then, since $|\tilde{\mathcal{C}}_a^{\mathcal{A}}| \leq |\tilde{\mathcal{C}}_0^{\mathcal{A}}|$ ([CR79], Theorem 9), $|\mathcal{C}_a^{\mathcal{A}}| \leq |\mathcal{C}_0^{\mathcal{A}}|$. Furthermore, from ([MR79], Corollary 2) $|\tilde{\mathcal{C}}_0^{\mathcal{A}}| \leq \frac{1}{n'+1}\left(2^{n'} + n' 2^{(n'-2)/3}\right)$ with equality if and only if $\mathcal{A}$ is an elementary Abelian 3-group. From ([MR79], Corollary 1), $|\tilde{\mathcal{C}}_0^{\mathcal{A}}| = \frac{2^{n'}}{n'+1}$ if and only if $n'+1$ is a power of 2. Multiplying $|\tilde{\mathcal{C}}_0^{\mathcal{A}}|$ by 2 and replacing $n = n'+1$ then gives the upper bound stated in the corollary. The lower bound follows from the observation that Construction E partitions the space $GF(2)^n$ into n binary single-grain codes and thus there exists a code with cardinality at least $2^n/n$. □

In [SR11], a single-grain code construction was given that produced codes of length $n = 2^m - 1$ with $\frac{2^n}{n+1} + 2^{\frac{(n-1)}{2}}$ codewords where m is a positive integer. In [MBK11], a single-grain code construction was enumerated that resulted in codes of length n where $n = 2^r$ (where r is a positive integer), that contained $\frac{2^n}{n}$ codewords.

Our construction extends for any n (via the set $\mathcal{A} = \mathbb{Z}_n$). When n is a power of 2, Construction E produces codes with the same cardinality as [MBK11]. Furthermore, for codes of length n where n is not a power of 2, Construction E provides codebooks with cardinalities strictly greater than $\frac{2^n}{n}$ by Corollary 16.

Since, for large n,

$$\frac{2^n}{n} > \frac{2^n}{n+1} + 2^{\frac{n-1}{2}},$$

Construction E improves upon the state of the art when n is not a power of 2 and $n \geq 15$.

In the next subsection, we provide a generalization of Construction E. We then derive constructions for single-grain codes that have larger cardinalities and extend the ideas to codes capable of correcting more than a single grain-error.

5.3.2 Improved grain codes using mappings

In [GSSSZ12], the authors make the observation that a single-asymmetric error-correcting code (and in particular a group code) can be constructed by defining a code over pairs of binary elements. Consider the map $\Gamma : \{0,1\}^2 \to GF(3)$, which is defined as follows:

$$(0,0) \to 0, (0,1) \to 1, (1,0) \to 2, (1,1) \to 0. \tag{5.7}$$

Note that the map is not one-to-one since both $(0,0)$ and $(1,1)$ map to 0. If the map Γ is applied to a binary vector of even length then it is simply applied to each pair of consecutive elements at a time (i.e., $\Gamma(0,1,0,0) = (\Gamma(0,1), \Gamma(0,0))$). Furthermore, if the Γ map is applied to a set of vectors it returns a set of ternary vectors that are the result of applying the map to each vector in the set. Using this map, codes that correct asymmetric errors were proposed

in [GSSSZ12]. In the following, we illustrate how to generalize the ideas from [GSSSZ12] (by using different mappings) to correct grain-errors.

Let $\mathcal{G}_{t,m} = (V, E)$ denote a simple graph (see Section 5.1.3) where $V = GF(2)^m$. That is, the vertices of $\mathcal{G}_{t,m}$ are the the vectors from $GF(2)^m$. For any $\boldsymbol{x}, \boldsymbol{y} \in V$, $(\boldsymbol{x}, \boldsymbol{y}) \in E$ if $\mathcal{B}_{t,M}(\boldsymbol{x}) \cap \mathcal{B}_{t,M}(\boldsymbol{y}) \neq \emptyset$. Recall from Section 5.1.3, a mapping $\Phi_{t,m} : GF(2)^m \to \{0, 1, \ldots, p-1\}$ is a p-coloring if it assigns different numbers to adjacent vertices. If the input to $\Phi_{t,m}$ is a vector of length mn, then the map is applied to each collection of m consecutive bits at a time. For example, if $m = 3$, then $\Phi_{t,3}(0, 0, 0, 1, 0, 1) = (\Phi_{t,3}(0, 0, 0)\ \Phi_{t,3}(1, 0, 1))$.

Construction F. *Suppose p is a prime number and $\Phi_{t,m} : GF(2)^m \to \{0, 1, \ldots, p-1\}$ is a p-coloring on $\mathcal{G}_{t,m}$. Let $\mathcal{C}_t$ be a t-random-error-correcting code over $GF(p)$ of length n. Then,*

$$\mathcal{C} = \{\boldsymbol{x} \in GF(2)^{mn} : \Phi_{t,m}(\boldsymbol{x}) \in \mathcal{C}_t\}. \tag{5.8}$$

Remark 6. *If $\mathcal{C}$ is a code created according to Construction F, then the map $\Phi_{t,m}$ can be interpreted as mapping the color classes of a p-coloring onto the symbols of a non-binary code $\mathcal{C}_t$. This interpretation will be useful in Section 5.5.*

Remark 7. *As noted in [GSSSZ12], since a code $\mathcal{C}$ created according to Construction E is a permutation of a group code, Construction E and Construction F coincide for the case where $p = 3$ and $\mathcal{C}_1$ (from (5.8) in Construction F) is a single random-error-correcting code over $GF(3)$.*

We now provide an example of a code created with Construction F.

Example 16. *Let the map Γ be as defined in (5.7). Note, from Lemma 32, that the map Γ is actually a coloring on $\mathcal{G}_{t,2}$ where the set of vectors $GF(2)^2$ are partitioned into color classes as follows:*

1. $\{(0\ 0), (1\ 1)\}$,

2. $\{(1\ 0)\}$,

3. $\{(0\ 1)\}$.

Let $\mathcal{C}_t$ be a t-random-error-correcting code over $GF(3)$ of length n. Then the set of vectors

$$\mathcal{C} = \{\boldsymbol{x} \in GF(2)^{2n} : \Gamma(\boldsymbol{x}) \in \mathcal{C}_t\} \tag{5.9}$$

is a code created according to Construction F.

Remark 8. *We note that when $\mathcal{C}_t$ is a single-random-error-correcting code, a code constructed according to Example 16 coincides with the ternary construction from [GSSSZ12] proposed in the context of asymmetric errors.*

We now prove that any code created according to Construction F is a t-mineral-error-correcting code.

Theorem 12. *Let $\mathcal{C}_t$ be a t-random-error-correcting code. Suppose $\mathcal{C}$ is a code created according to Construction F with $\mathcal{C}_t$ as the constituent code. Then, $\mathcal{C}$ is a t-mineral-error-correcting code.*

Proof. The result will be proven by showing that for any codewords $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{C}$ where $\boldsymbol{x} \neq \boldsymbol{y}$, $\mathcal{B}_{t,M}(\boldsymbol{x}) \cap \mathcal{B}_{t,M}(\boldsymbol{y}) = \emptyset$. Consider two codewords $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{C}$ such that $\boldsymbol{x} \neq \boldsymbol{y}$. There are two cases to consider: either 1) $\Phi_{t,m}(\boldsymbol{x}) = \Phi_{t,m}(\boldsymbol{y})$ or 2) $\Phi_{t,m}(\boldsymbol{x}) \neq \Phi_{t,m}(\boldsymbol{y})$. Recall that, by construction, $\Phi_{t,m}(\boldsymbol{x}), \Phi_{t,m}(\boldsymbol{y}) \in \mathcal{C}_t$.

Suppose $\Phi_{t,m}(\boldsymbol{x}) = \Phi_{t,m}(\boldsymbol{y})$. Then, since $\boldsymbol{x} \neq \boldsymbol{y}$, there exists an index i where $1 \leq i \leq n$ such that $\Phi_{t,m}(x_{(i-1)m+1}, \ldots, x_{im}) = \Phi_{t,m}(y_{(i-1)m+1}, \ldots, y_{im})$ but $(x_{(i-1)m+1}, \ldots, x_{im}) \neq (y_{(i-1)m+1}, \ldots, y_{im})$. For shorthand, let $\boldsymbol{v}_1 = (x_{(i-1)m+1}, \ldots, x_{im})$ and $\boldsymbol{v}_2 = (y_{(i-1)m+1}, \ldots, y_{im})$. Since $\Phi_{t,m}(\boldsymbol{v}_1) = \Phi_{t,m}(\boldsymbol{v}_2)$, the vectors $\boldsymbol{v}_1, \boldsymbol{v}_2$ map to the same color class under $\Phi_{t,m}$, which implies that $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ are not adjacent in $\mathcal{G}_{t,m}$. By definition, if $\boldsymbol{v}_1, \boldsymbol{v}_2$ are not adjacent in $\mathcal{G}_{t,m}$, $\mathcal{B}_{t,M}(\boldsymbol{v}_1) \cap \mathcal{B}_{t,M}(\boldsymbol{v}_2) = \emptyset$. Thus, for any t-mineral-errors (of length m) $\boldsymbol{e}_{\boldsymbol{v}_1}, \boldsymbol{e}_{\boldsymbol{v}_2}$, we

have $\boldsymbol{v}_1 + \boldsymbol{e}_{\boldsymbol{v}_1} \neq \boldsymbol{v}_2 + \boldsymbol{e}_{\boldsymbol{v}_2}$. Then, there do not exist any t-mineral-errors $\boldsymbol{e}_{\boldsymbol{x}}, \boldsymbol{e}_{\boldsymbol{y}}$ such that $\boldsymbol{x} + \boldsymbol{e}_{\boldsymbol{x}} = \boldsymbol{y} + \boldsymbol{e}_{\boldsymbol{y}}$. Thus, $\mathcal{B}_{t,M}(\boldsymbol{x}) \cap \mathcal{B}_{t,M}(\boldsymbol{y}) = \emptyset$.

Suppose now that $\Phi_{t,m}(\boldsymbol{x}) \neq \Phi_{t,m}(\boldsymbol{y})$. Then, since $\Phi_{t,m}(\boldsymbol{x}), \Phi_{t,m}(\boldsymbol{y}) \in \mathcal{C}_t$, there exists a set of at least $2t+1$ indices from $\{1, 2, \ldots, n\}$, denoted as $\mathcal{I}$, such that $\forall j \in \mathcal{I}$, $\Phi_{t,m}(x_{(j-1)m+1}, \ldots, x_{jm}) \neq \Phi_{t,m}(y_{(j-1)m+1}, \ldots, y_{jm})$. Since

$$\Phi_{t,m}(x_{(j-1)m+1}, \ldots, x_{jm}) \neq \Phi_{t,m}(y_{(j-1)m+1}, \ldots, y_{jm}),$$

$d_H((x_{(j-1)m+1}, \ldots, x_{jm}), (y_{(j-1)m+1}, \ldots, y_{jm})) \geq 1$ for every $j \in \mathcal{I}$ and so $d_H(\boldsymbol{x}, \boldsymbol{y}) \geq 2t+1$. Thus, $\mathcal{B}_{t,R}(\boldsymbol{x}) \cap \mathcal{B}_{t,R}(\boldsymbol{y}) = \emptyset$ where $\mathcal{B}_{t,R}$ denotes the error-ball for t random-errors (as discussed in Section 5.1.1). From Claim 18, then $\mathcal{B}_{t,M}(\boldsymbol{x}) \cap \mathcal{B}_{t,M}(\boldsymbol{y}) = \emptyset$ as well and the proof is complete. □

Notice that according to Theorem 12, the code from Example 16 is a t-mineral-error-correcting code. Corollary 17 follows from Claim 19.

Corollary 17. *Let $\mathcal{C}'$ be a t-mineral-error-correcting code of length mn created according to Construction F. Then,*

$$\mathcal{C} = \{\boldsymbol{x} \in GF(2)^{mn+1} : (x_2, \ldots, x_{mn+1}) \in \mathcal{C}'\}$$

is a t-grain-error-correcting code.

Although Construction F provides a method to construct t-mineral-error-correcting codes, it is not straightforward to compute the sizes of the resulting codes because the color classes of the map $\Phi_{t,m}$ are not always of the same size. As a starting point, in this subsection we only consider single-mineral codes created using Construction F with the map Γ as described in Example 16. Even with the simple map Γ, computing the cardinalities of the resulting codes from Construction F is not straightforward. In the following subsection, we analyze the codes from Example 16 for arbitrary t.

Table 5.2: Upper and Lower Bounds for single grain-error-correcting Codes

Length	Previous Lower Bound	Current Lower Bound	Upper Bound
3	4 [SR11]	4 [SR11]	4 [SR11]
4	6 [SR11]	6 [SR11]	6 [SR11]
5	8 [SR11]	8 [SR11]	8 [SR11]
6	16 [SR11]	16 [SR11]	16 [SR11]
7	26 [SR11]	26 [SR11]	26 [SR11]
8	44[SR11]	44 [SR11]	44 [SR11]
9	44 [SR11]	**64**	112
10	64 [MBK11]	**110**	204
11	128 [MBK11]	**210**	372
12	256 [MBK11]	**360**	682
13	512 [MBK11]	**702**	1260
14	1024 [MBK11]	**1200**	2340
15	2176 [SR11]	**2400**	4368
16	4096 [MBK11]	**4096**	8190
17	4096 [MBK11]	**7712**	15420
18	8192 [MBK11]	**14592**	29126
19	16384 [MBK11]	**27596**	55188
20	32768 [MBK11]	**52432**	104856

Recall that from Remark 8, the single asymmetric error-correcting codes proposed in [GSSSZ12] (using the ternary construction) are a special case of Construction F. Therefore, the codes from (Table II, column 4, [GSSSZ12]) are single-mineral codes. Therefore, we can obtain new single-grain codes by appending an information bit to these codes. The cardinalities displayed in the column titled 'Current Lower Bound' (second column) of Table 5.2 (shown below) for $9 \leq n \leq 15$ are the result of this operation. Note that the codes enumerated from [GSSSZ12] were the result of a computerized search and to limit the search space, the search was only carried out on codes of length at most 15. For $n \geq 16$ the cardinalities in the second column of Table 5.2 (marked in bold) can be obtained from Construction E using the group codes found in Table 1 in [CR79]. The first column in Table 5.2 shows the cardinalities of the largest possible codebooks using constructions from [MBK11] and [SR11]. The third column in the table is the upper bound from Corollary 12 (Section 5.2), which can also be found in [KZ13].

5.3.3 Multiple grain-error codes using the Γ coloring

In this subsection, multiple grain-error-correcting codes are studied. In particular, we consider an alternative interpretation of the codes from Example 16. Using this interpretation, we derive a lower bound on the size of a mineral code created according to Example 16 for the case where the code $\mathcal{C}_t$ is linear.

Notice that if the Hamming weight enumerator for the constituent code $\mathcal{C}_t$ in Example 16 is given, then the size of the code $\mathcal{C}$ can be expressed as a function of the Hamming weight enumerator for $\mathcal{C}_t$. We denote the Hamming weight enumerator of a code $\mathcal{C}$ as $W_{\mathcal{C}}(x,z) = \sum_{i=0}^{n} W_{i,n-i} z^i x^{n-i}$ where $W_{i,n-i}$ represents the number of codewords in $\mathcal{C}$ whose Hamming weight is i. The following lemma is similar to Theorem 9 in [GSSSZ12] and so the proof is omitted.

Lemma 39. *Let $\mathcal{C}_t$ be a ternary code of length n used in Example 16 with Hamming weight enumerator*

$$\mathcal{W}_{\mathcal{C}_t}(x,z) = \sum_{i=0}^{n} W_{i,n-i} z^i x^{n-i}.$$

Then, the resulting mineral-error-correcting code $\mathcal{C}$ has cardinality $|\mathcal{C}| = W_{\mathcal{C}_t}(2,1)$. Prepending an additional information bit to every codeword in $\mathcal{C}$ results in a grain-error-correcting code with cardinality $2|\mathcal{C}|$.

Remark 9. *Note that in general the weight enumerator for any t-random-error-correcting ternary code $\mathcal{C}_t$ is not necessarily known.*

Using Lemma 39, the cardinalities of grain codes created according to Example 16 with odd lengths between 11 and 29 are displayed in Table 5.3. Each entry consists of 3 numbers. For instance, in Table 5.3, there are 3 numbers $(3, 68, 168)$ under the entry for length 11 and $t = 2$. Since for $1 < t < n$ there are no existing grain-error-correcting codebooks to compare with, we naively constructed a t-grain-error-correcting code by prepending an additional information bit to the start of a t-random-error-correcting code.

The first entry (from each triplet) in Table 5.3 is the cardinality of the largest linear t-random-error-correcting binary code found in [MAG11] of length $n-1$ prepended by an additional information bit. The second entry is the cardinality of a code created from Example 16. This number was computed from the known weight enumerators of the largest known linear ternary codes from [MAG11] prepended by an additional information bit. The third entry is the non-asymptotic upper bound from Theorem 10 and Lemma 33.

In the following, we provide a variation of the codes from Example 16 in order to provide an explicit lower bound on the size of codes created as in Example 16 when $\mathcal{C}_t$ is linear. This will be studied in more detail in Section 5.4.

Construction G. *Let r, ℓ be positive integers where $r \leq \ell$. Let $H' = (\boldsymbol{h}'_1, \ldots, \boldsymbol{h}'_\ell)$ be an $r \times \ell$ parity check matrix of a ternary code $\mathcal{C}'$ of length ℓ that can correct up to t random-errors (where each $\boldsymbol{h}'_i$ represents the ith column in H', $1 \leq i \leq \ell$). Let H be an $r \times 2\ell$ ternary matrix,*

$$H = (\boldsymbol{h}_1, \ldots, \boldsymbol{h}_{2\ell}) = (2\boldsymbol{h}'_1, \boldsymbol{h}'_1, 2\boldsymbol{h}'_2, \boldsymbol{h}'_2, \ldots, 2\boldsymbol{h}'_\ell, \boldsymbol{h}'_\ell).$$

Let $\boldsymbol{a}$ be an arbitrary element in $GF(3)^r$. Then,

$$\mathcal{C}_{\boldsymbol{a}} = \{\boldsymbol{x} \in GF(2)^{2\ell} : H\boldsymbol{x} = \boldsymbol{a}\}, \tag{5.10}$$

where the vector operations are performed in the vector space $GF(3)^r$.

The following lemma will be useful in proving the correctness of Construction G.

Lemma 40. *Let r, ℓ be positive integers where $r \leq \ell$ and let the matrices H', H be as in Construction G. Then for any $\boldsymbol{x} \in GF(2)^{2\ell}$, $H \cdot \boldsymbol{x} = H' \cdot \Gamma(\boldsymbol{x})$.*

Proof. For any $\boldsymbol{x} = (x_1, \ldots, x_{2\ell}) \in GF(2)^{2\ell}$ we have $H \cdot \boldsymbol{x} = \sum_{i=1}^{2\ell} \boldsymbol{h}_i \cdot x_i$ where $\boldsymbol{h}_i \in GF(3)^r$.

Table 5.3: Cardinalities of Grain-error-correcting Codes

Length	$t=2$			$t=3$			$t=4$			$t=5$		
11	16	68	168	-	-	-	-	-	-	-	-	-
13	32	132	476	-	-	-	-	-	-	-	-	-
15	128	312	1406	32	260	800	-	-	-	-	-	-
17	512	836	4302	64	516	2132	-	-	-	-	-	-
19	1024	2636	13560	256	1028	5892	16	1028	3854	-	-	-
21	4096	9376	43804	1024	2144	16836	64	2052	9878	-	-	-
23	16384	35648	144380	4096	4688	49572	128	4100	26100	32	4100	18740
25	32768	49072	483954	8192	8896	149804	256	8320	71018	64	8196	46762
27	131072	190912	1645392	16384	20808	463074	1024	17216	198660	256	16388	119626
29	524288	747520	5662422	32768	53460	1459848	2048	34096	570038	512	32772	313846

Consider the quantity

$$\begin{aligned} H \cdot \boldsymbol{x} &= \sum_{i=1}^{2\ell} \boldsymbol{h}_i \cdot x_i \\ &= \sum_{j=1, j \text{ odd}}^{2\ell-1} (\boldsymbol{h}_j, \boldsymbol{h}_{j+1}) \cdot (x_j, x_{j+1})^T \\ &= \sum_{j=1, j \text{ odd}}^{2\ell-1} (2\boldsymbol{h}'_{\lceil \frac{j}{2} \rceil}, \boldsymbol{h}'_{\lceil \frac{j}{2} \rceil}) \cdot (x_j, x_{j+1})^T. \end{aligned} \tag{5.11}$$

There are the 4 possibilities for (x_j, x_{j+1}):

1. $(x_j, x_{j+1}) = (0, 0)$,

2. $(x_j, x_{j+1}) = (0, 1)$,

3. $(x_j, x_{j+1}) = (1, 0)$,

4. $(x_j, x_{j+1}) = (1, 1)$.

If any of conditions $1) - 4)$ hold, then it can be verified that when j is odd, we have (where Γ is as defined in (5.7))

$$(2\boldsymbol{h}'_{\lceil \frac{j}{2} \rceil}, \boldsymbol{h}'_{\lceil \frac{j}{2} \rceil}) \cdot (x_j, x_{j+1})^T = \boldsymbol{h}'_{\lceil \frac{j}{2} \rceil} \cdot \Gamma(x_j, x_{j+1}).$$

Then, continuing from (5.11),

$$
\begin{aligned}
H \cdot \boldsymbol{x} &= \sum_{j=1, j \text{ odd}}^{2\ell-1} (2\boldsymbol{h}'_{\lceil \frac{j}{2} \rceil}, \boldsymbol{h}'_{\lceil \frac{j}{2} \rceil}) \cdot (x_j, x_{j+1})^T \\
&= \sum_{j=1, j \text{ odd}}^{2\ell-1} \boldsymbol{h}'_{\lceil \frac{j}{2} \rceil} \cdot \Gamma(x_j, x_{j+1}) \\
&= \sum_{k=1}^{\ell} \boldsymbol{h}'_k \cdot \Gamma(x_{2k-1}, x_{2k}) \\
&= H' \cdot \Gamma(\boldsymbol{x}).
\end{aligned}
$$

□

We now prove the correctness of Construction G.

Theorem 13. *Suppose $\mathcal{C}_{\boldsymbol{a}}$ is a code created according to Construction G. Then, $\mathcal{C}_{\boldsymbol{a}}$ is a t-mineral-error-correcting code.*

Proof. Let H' be a parity check matrix of dimension r (where $r \leq \ell$) for the code $\mathcal{C}'$ of length ℓ that can correct up to t random-errors. For any $\boldsymbol{a} \in \mathcal{A}$, let $\mathcal{C}'_{\boldsymbol{a}} = \{\boldsymbol{x} \in GF(3)^\ell : H' \cdot \boldsymbol{x} = \boldsymbol{a}\}$. Notice that for any $\boldsymbol{a} \in \mathcal{A}$, $\mathcal{C}'_{\boldsymbol{a}}$ is a ternary t-random-error-correcting code. Recall from Construction G that $\mathcal{C}_{\boldsymbol{a}} = \{\boldsymbol{x} \in GF(2)^{2\ell} : H\boldsymbol{x} = \boldsymbol{a}\}$ where $H = (2\boldsymbol{h}'_1, \boldsymbol{h}'_1, 2\boldsymbol{h}'_2, \boldsymbol{h}'_2, \ldots, 2\boldsymbol{h}_\ell, \boldsymbol{h}'_\ell) = (\boldsymbol{h}_1, \ldots, \boldsymbol{h}_{2\ell})$ (and each $\boldsymbol{h}_i, \boldsymbol{h}'_j$ denotes a column in H or H', respectively for $1 \leq i \leq 2\ell$ and $1 \leq j \leq \ell$).

From Lemma 40, for any vector $\boldsymbol{x} \in GF(2)^n$, $H \cdot \boldsymbol{x} = H' \cdot \Gamma(\boldsymbol{x})$. Therefore, it follows that $H \cdot \boldsymbol{x} = \boldsymbol{a}$ if and only if $H' \cdot \Gamma(\boldsymbol{x}) = \boldsymbol{a}$. Then, we can write $\mathcal{C}_{\boldsymbol{a}} = \{\boldsymbol{x} \in GF(2)^n : \Gamma(\boldsymbol{x}) \in \mathcal{C}'_{\boldsymbol{a}}\}$. Since $\mathcal{C}'_a$ is a t-random-error-correcting code, $\mathcal{C}_{\boldsymbol{a}}$ is a t-mineral-error-correcting code by Example 16 and Theorem 12. □

Using the interpretation of the codes from Example 16 provided by Construction G, we now state a simple lower bound on the size of a code created as in Example 16. Recall from Theorem 13, Construction G is a special case of the codes from Example 16. The lower bound in Corollary 18 will be improved in the next section.

Recall for the following corollary that $\mathcal{A}$ denotes an Abelian group.

Corollary 18. *Let $\mathcal{C}'$ be a t-random-error-correcting ternary code of length $\ell = \frac{n}{2}$ (where n is even) with a parity check matrix H' of dimension r. Then there exists an $\boldsymbol{a} \in \mathcal{A}$, such that the code $\mathcal{C}_{\boldsymbol{a}}$ created according to Construction G of length n with the constituent code $\mathcal{C}'$ satisfies $|\mathcal{C}_{\boldsymbol{a}}| \geq \frac{2^n}{3^r}$.*

Proof. Notice that each of the $2^{2\ell}$ vectors from $GF(2)^{2\ell}$ will map to exactly one code $\mathcal{C}_{\boldsymbol{a}}$ as in (5.10). Thus, the matrix H partitions the space $GF(2)^{2\ell}$ into $|\mathcal{A}|$ non-overlapping codes $\mathcal{C}_{\boldsymbol{a}_1}, \mathcal{C}_{\boldsymbol{a}_2}, \mathcal{C}_{\boldsymbol{a}_3}, \ldots, \mathcal{C}_{\boldsymbol{a}_{3^r}}$ where each $\boldsymbol{a}_i \in \mathcal{A}$ for $1 \leq i \leq 3^r$. By the pigeonhole principle, there must exist a code with cardinality at least $\frac{2^{2\ell}}{|\mathcal{A}|} = \frac{2^n}{3^r}$. □

Recall, Construction G was introduced as a tool that can be used to provide lower bounds on the sizes of codes created according to the more general Construction F. Although the lower bound from Lemma 39 can be used, Lemma 39 has the potential drawback that it requires knowledge of the weight enumerator for a non-binary code which may not be known. The purpose of the next section is to use Construction G to produce a lower bound that holds for general n. The lower bound in the next section only requires the knowledge of the number of parity symbols for the non-binary constituent code. It will be demonstrated in Table 5.4 that in many cases the resulting lower bound guarantees codebooks with strictly more codewords than the largest known binary codebooks. In the next section, we use Fourier analysis to improve the lower bound on $\mathcal{C}_{\boldsymbol{a}}$ from Construction G.

5.4 An Improvement on the lower bounds on the cardinality of grain and mineral codes when $t \geq 2$

In this section, we improve the lower bound from the previous section for the cardinality of a t-mineral-error-correcting code created according to Construction G. The approach will be similar to [MR79], where the cardinalities of the Constantin-Rao codes [CR79] were derived using discrete Fourier analysis.

Let $\mathcal{A}$ be the additive Abelian group of $GF(3)^r$. Let $\mathcal{C}_{\boldsymbol{a}}$ denote a code created using Construction G where as before $\boldsymbol{a}$ is an element from $\mathcal{A}$ used in the construction. Suppose further that $\mathcal{C}'$ is a ternary code of length ℓ with a parity check matrix H' that can correct up to t random-errors where $\mathcal{C}'$ is the constituent code used in Construction G. For $1 \leq i \leq \ell$, recall from the construction that $\boldsymbol{h}'_i$ refers to the ith column of H' and that for $1 \leq j \leq 2\ell$, $\boldsymbol{h}_j$ refers to the jth column of H where $H = (\boldsymbol{h}_1, \ldots, \boldsymbol{h}_{2\ell}) = (2\boldsymbol{h}'_1, \boldsymbol{h}'_1, 2\boldsymbol{h}'_2, \boldsymbol{h}'_2, \ldots, 2\boldsymbol{h}'_\ell, \boldsymbol{h}'_\ell)$.

For $\boldsymbol{x} = (x_1, \ldots, x_{2\ell}) \in GF(2)^{2\ell}$, consider the mapping $\gamma : GF(2)^{2\ell} \to \mathcal{A}$ defined as

$$\gamma(\boldsymbol{x}) = H \cdot \boldsymbol{x} = \sum_{j=1}^{2\ell} x_j \boldsymbol{h}_j = \sum_{i=1}^{\ell} x_{2i} \boldsymbol{h}'_i + \sum_{k=1}^{\ell} 2x_{2k-1} \boldsymbol{h}'_k. \tag{5.12}$$

In order to compute $|\mathcal{C}_{\boldsymbol{a}}|$, we count the number of times each element $\boldsymbol{a} \in \mathcal{A}$ is covered by some vector $\boldsymbol{x} \in GF(2)^{2\ell}$ through γ. Let $f : \mathcal{A} \to \mathbb{N}$ where

$$f(\boldsymbol{a}) = |\{\boldsymbol{x} \in GF(2)^{2\ell} : \gamma(\boldsymbol{x}) = H \cdot \boldsymbol{x} = \boldsymbol{a}\}|. \tag{5.13}$$

We state the following claim for clarity. Recall, $M(n, t)$ refers to the maximum size of a t-grain-error-correcting code of length n.

Claim 23. *Let n, ℓ be positive integers such that $n = 2\ell + 1$. Let $\mathcal{C}_{\boldsymbol{a}}$ be a code of length 2ℓ created according to Construction G where $\boldsymbol{a} \in \mathcal{A}$. Then, $|\mathcal{C}_{\boldsymbol{a}}| = f(\boldsymbol{a})$ and $M(n, t) \geq 2|\mathcal{C}_{\boldsymbol{a}}| = 2f(\boldsymbol{a})$.*

We are now ready to derive lower bounds on the sizes of codes created from Construction G using Fourier analysis. The following lemma will be used in the proof of Theorem 14. Recall from Section 5.1.5, for $\boldsymbol{a}, \boldsymbol{b} \in \mathcal{A}$,

$$< \boldsymbol{a}, \boldsymbol{b} >= \prod_{i=1}^{r} (\zeta_3)^{a_i b_i} = (\zeta_3)^{\sum_{i=1}^{r} a_i b_i} = (\zeta_3)^{\boldsymbol{a}^T \cdot \boldsymbol{b}}$$

where ζ_3 is a third root of unity. In the remainder, for some positive integer k, $(\zeta_3)^k$ will be written as ζ_3^k.

In the next lemma, we make use of the following function. Let $F : \mathcal{A} \times \mathcal{A} \to \mathbb{C}$ where for $\boldsymbol{a} \in \mathcal{A}, \boldsymbol{b} \in \mathcal{A}$,

$$F(\boldsymbol{a}, \boldsymbol{b}) = 1 + < -\boldsymbol{a}, \boldsymbol{b} > + < -\boldsymbol{a}, 2\boldsymbol{b} > + < -\boldsymbol{a}, \boldsymbol{b} >< -\boldsymbol{a}, 2\boldsymbol{b} > .$$

Lemma 41. *For any* $\boldsymbol{a}, \boldsymbol{b} \in \mathcal{A}$,

$$F(\boldsymbol{a}, \boldsymbol{b}) = \begin{cases} 4 & \text{if } \boldsymbol{a}^T \cdot \boldsymbol{b} = \sum_{i=1}^{r} a_i b_i \equiv 0 \mod 3, \text{ and} \\ 1 & \text{otherwise.} \end{cases}$$

Proof. First consider the case where $\boldsymbol{a}^T \cdot \boldsymbol{b} \equiv 0 \mod 3$. Notice that if $\boldsymbol{a}^T \cdot \boldsymbol{b} \equiv 0 \mod 3$, then $< \boldsymbol{a}, \boldsymbol{b} >= 1$. Since $\boldsymbol{a}^T \cdot \boldsymbol{b} \equiv 0 \mod 3$ we have $-\boldsymbol{a}^T \cdot \boldsymbol{b} = -\boldsymbol{a}^T \cdot 2\boldsymbol{b} \equiv 0 \mod 3$ and so the quantity in the Lemma is equal to 4.

Consider the case now where $\boldsymbol{a}^T \cdot \boldsymbol{b} \not\equiv 0 \mod 3$. Recall ζ_3 is a cubic root of unity and note that $< -\boldsymbol{a}, 2\boldsymbol{b} >=< -\boldsymbol{a}, \boldsymbol{b} >^2$. Then,

$$\begin{aligned} & < -\boldsymbol{a}, \boldsymbol{b} > + < -\boldsymbol{a}, 2\boldsymbol{b} > + < -\boldsymbol{a}, \boldsymbol{b} >< -\boldsymbol{a}, 2\boldsymbol{b} > \\ & =< -\boldsymbol{a}, \boldsymbol{b} > + < -\boldsymbol{a}, \boldsymbol{b} >^2 + < -\boldsymbol{a}, \boldsymbol{b} >^3 \\ & = \zeta_3 + \zeta_3^2 + \zeta_3^3 \\ & = 0, \end{aligned}$$

and so $F(\boldsymbol{a}, \boldsymbol{b}) = 1$. □

Given an input $\boldsymbol{c} \in \mathcal{A}$, let $\beta : \mathcal{A} \to \{0, \ldots, \ell\}$ be defined as follows

$$\beta(\boldsymbol{c}) = |\{1 \leq i \leq \ell : \boldsymbol{c}^T \cdot \boldsymbol{h}'_i \equiv 0 \mod 3\}| \tag{5.14}$$

where $\boldsymbol{h}'_i$ refers to the i-th column of H'.

Table 5.4: Comparison of sizes of grain-error-correcting codes with the lower bound from Corollary 20

Length	$t=2$		$t=3$		$t=4$	
11	68	34	-	-	-	-
13	132	46	-	-	-	-
15	312	148	260	64	-	-
17	836	552	516	86	-	-
19	2636	2170	1028	116	1028	116
21	9376	8642	2144	356	2052	156
23	35648	34534	4688	1314	4100	208
25	49072	46044	8896	1754	8320	634
27	190912	184130	20808	6868	17216	2340
29	747520	736466	53460	27324	34096	3122

The following function will be used in the proof of Theorem 14. Let $I : GF(2)^{2\ell} \times \mathcal{A} \to \{0,1\}$ denote the indicator function where for $\boldsymbol{x} \in GF(2)^{2\ell}$ and $\boldsymbol{a} \in \mathcal{A}$,

$$I(\boldsymbol{x}, \boldsymbol{a}) = \begin{cases} 1 & \text{if } \gamma(\boldsymbol{x}) = \boldsymbol{a}, \\ 0 & \text{otherwise} \end{cases} \tag{5.15}$$

where γ is as defined in (5.12).

We are now ready for the main result of this section.

Theorem 14. *For any* $\boldsymbol{b} \in \mathcal{A}$, $f(\boldsymbol{b}) = \frac{1}{3^r} \sum_{\boldsymbol{a} \in \mathcal{A}} < \boldsymbol{b}, \boldsymbol{a} > 4^{\beta(\boldsymbol{a})}$.

Proof. Consider $\boldsymbol{c} \in \mathcal{A}$. As in [MR79], we proceed by computing the Fourier transform $\hat{f}(\boldsymbol{c})$ (as defined as in Section 5.1.5). First note that from (5.15), we can write $f(\boldsymbol{a}) =$

$\sum_{\boldsymbol{x}\in GF(2)^{2\ell}} I(\boldsymbol{x},\boldsymbol{a})$ where $\boldsymbol{a}\in\mathcal{A}$. We have

$$\begin{aligned}\hat{f}(\boldsymbol{c}) &= \sum_{\boldsymbol{a}\in\mathcal{A}} <\boldsymbol{c},-\boldsymbol{a}> f(\boldsymbol{a})\\ &= \sum_{\boldsymbol{a}\in\mathcal{A}} <-\boldsymbol{c},\boldsymbol{a}> f(\boldsymbol{a})\\ &= \sum_{\boldsymbol{a}\in\mathcal{A}} <-\boldsymbol{c},\boldsymbol{a}> \sum_{x\in GF(2)^{2\ell}} I(\boldsymbol{x},\boldsymbol{a})\\ &= \sum_{\boldsymbol{a}\in\mathcal{A}}\sum_{\boldsymbol{x}\in GF(2)^{2\ell}} <-\boldsymbol{c},\boldsymbol{a}> I(\boldsymbol{x},\boldsymbol{a})\\ &= \sum_{\boldsymbol{x}\in GF(2)^{2\ell}}\sum_{\boldsymbol{a}\in\mathcal{A}} <-\boldsymbol{c},\boldsymbol{a}> I(\boldsymbol{x},\boldsymbol{a}).\end{aligned}$$

Note that for a fixed $\boldsymbol{x}\in GF(2)^{2\ell}$, $\sum_{\boldsymbol{a}\in\mathcal{A}} <-\boldsymbol{c},\boldsymbol{a}> I(\boldsymbol{x},\boldsymbol{a}) =<-\boldsymbol{c},\gamma(\boldsymbol{x})>$. Then,

$$\begin{aligned}\hat{f}(\boldsymbol{c}) &= \sum_{\boldsymbol{x}\in GF(2)^{2\ell}} <-\boldsymbol{c},\gamma(\boldsymbol{x})>\\ &= \sum_{\boldsymbol{x}\in GF(2)^{2\ell}} <-\boldsymbol{c},x_1\boldsymbol{h}_1+\cdots+x_{2\ell}\boldsymbol{h}_{2\ell}>\\ &= \sum_{\boldsymbol{x}\in GF(2)^{2\ell}} <-\boldsymbol{c},x_1\boldsymbol{h}_1>\cdots<-\boldsymbol{c},x_{2\ell}\boldsymbol{h}_{2\ell}>,\end{aligned}$$

where the last equality follows from the property that for $\boldsymbol{a}_1,\boldsymbol{a}_2,\boldsymbol{a}_3\in\mathcal{A}$, $<-\boldsymbol{a}_1,\boldsymbol{a}_2+\boldsymbol{a}_3>=<-\boldsymbol{a}_1,\boldsymbol{a}_2><-\boldsymbol{a}_1,\boldsymbol{a}_3>$.

Notice that each x_i is equal to either 0 or 1 (where $1\le i\le 2\ell$). If $x_i=0$, then clearly $<-\boldsymbol{c},x_i\boldsymbol{h}_i>=1$. If $x_i=1$, $<-\boldsymbol{c},x_i\boldsymbol{h}_i>=<-\boldsymbol{c},\boldsymbol{h}_i>$. Thus, by suitably collecting terms (and by induction on ℓ), we can write

$$\hat{f}(\boldsymbol{c}) = \prod_{i=1}^{2\ell}(1+<-\boldsymbol{c},\boldsymbol{h}_i>).$$

Let j be an integer such that $1\le j\le\ell$. Then from the definition of H (see also (5.12)) we can write $(1+<-\boldsymbol{c},\boldsymbol{h}_{2j}>)(1+<-\boldsymbol{c},\boldsymbol{h}_{2j-1}>) = (1+<-\boldsymbol{c},\boldsymbol{h}'_j>)(1+<-\boldsymbol{c},2\boldsymbol{h}'_j>)$.

Thus, we can rewrite $\hat{f}(\boldsymbol{c})$ in terms of the $\boldsymbol{h}'_i$ terms so that

$$\begin{aligned}\hat{f}(\boldsymbol{c}) &= \prod_{i=1}^{\ell}(1+ < -\boldsymbol{c}, \boldsymbol{h}'_i > + < -\boldsymbol{c}, 2\boldsymbol{h}'_i > + \\ &\qquad\qquad < -\boldsymbol{c}, \boldsymbol{h}'_i >< -\boldsymbol{c}, 2\boldsymbol{h}'_i >) \\ &= \prod_{i=1}^{\ell} F(\boldsymbol{c}, \boldsymbol{h}'_i) \\ &= 4^{\beta(\boldsymbol{c})}.\end{aligned}$$

The equality follows from Lemma 41. Recall, from Section 5.1.5 that the inverse Fourier transform of $\hat{f}$ is $f(\boldsymbol{b}) = \frac{1}{3^r}\sum_{\boldsymbol{a}\in\mathcal{A}} < \boldsymbol{a}, \boldsymbol{b} > \hat{f}(\boldsymbol{a})$. Thus, since $\hat{f}(\boldsymbol{a}) = 4^{\beta(\boldsymbol{a})}$, we have that for an element $\boldsymbol{b} \in \mathcal{A}$,

$$\begin{aligned}f(\boldsymbol{b}) &= \frac{1}{3^r}\sum_{\boldsymbol{a}\in\mathcal{A}} < \boldsymbol{a}, \boldsymbol{b} > \hat{f}(\boldsymbol{a}) \\ &= \frac{1}{3^r}\sum_{\boldsymbol{a}\in\mathcal{A}} < \boldsymbol{a}, \boldsymbol{b} > 4^{\beta(\boldsymbol{a})}.\end{aligned}$$

□

Corollary 19. *For any $\boldsymbol{b} \in \mathcal{A}$, $f(\boldsymbol{b}) \leq f(\boldsymbol{0})$.*

Proof. As in [MR79], this is because for any $\boldsymbol{a}, \boldsymbol{b} \in \mathcal{A}$, $| < \boldsymbol{a}, \boldsymbol{b} > | \leq 1$. Thus, $f(\boldsymbol{b}) = \frac{1}{3^r}\sum_{\boldsymbol{a}\in\mathcal{A}} < \boldsymbol{b}, \boldsymbol{a} > 4^{\beta(\boldsymbol{a})} \leq \frac{1}{3^r}\sum_{\boldsymbol{a}\in\mathcal{A}} 4^{\beta(\boldsymbol{a})} = f(\boldsymbol{0})$. □

Thus, choosing $\boldsymbol{a} = \boldsymbol{0}$ in Construction G maximizes the cardinality of the resulting code. The following lemma is another consequence of Theorem 14.

Lemma 42. *For positive integers r, ℓ where $r \leq \ell$, $f(\boldsymbol{0}) \geq \frac{4^\ell}{3^r} + 2\left(\frac{4}{3}\right)^r - 2 \cdot \frac{4}{3}$.*

Proof. From Theorem 14, we have that $f(\boldsymbol{0}) = \frac{1}{3^r}\sum_{\boldsymbol{a}\in\mathcal{A}} 4^{\beta(\boldsymbol{a})}$. Clearly, $\beta(\boldsymbol{0}) = \ell$ and so $f(\boldsymbol{0}) = \frac{1}{3^r}\left(4^\ell + \sum_{\boldsymbol{a}\in\mathcal{A}, \boldsymbol{a}\neq\boldsymbol{0}} 4^{\beta(\boldsymbol{a})}\right)$. We define the sets $\mathcal{T}_0 = \{\boldsymbol{0}\}, \mathcal{N}_0 = \{\boldsymbol{0}\}$, and $\mathcal{N}'_0 = \{\boldsymbol{0}\}$.

In the following we define the sets $\mathcal{N}_j, \mathcal{N}'_j$, and $\mathcal{T}_j$ recursively (starting at $j = 1$) where j is an integer such that $1 \leq j \leq r-1$. Consider the sub-matrix H'_j consisting of the first $r-j$

columns of H' where H' is the parity check matrix for $\mathcal{C}'$ with columns h'_i and $1 \leq i \leq \ell$. Let $\mathcal{N}_j = \{\boldsymbol{g} \in \mathcal{A} : \boldsymbol{g}^T \cdot H'_j = \mathbf{0}\}$. Notice that since $H'_j \in r \times \ell$ has rank at most $r - j$, $|\mathcal{N}_j| \geq 3^j$. Let $\mathcal{N}'_j \subseteq \mathcal{N}_j$ be chosen so that $|\mathcal{N}'_j| = 3^j$ and $\mathcal{N}'_j \subset N'_{j-1} \cdots \subset N_0$. Let $\mathcal{T}_j = \mathcal{N}'_j \setminus \mathcal{N}'_{j-1}$. Under this setup, for any $0 \leq k < j$, $\mathcal{T}_j \cap \mathcal{T}_k = \emptyset$. Now, for any $\mathcal{T}_j$, we have

$$|\mathcal{T}_j| = |\mathcal{N}'_j| - |\mathcal{N}'_{j-1}| = 3^j - 3^{j-1}.$$

Notice that for any $\boldsymbol{u} \in \mathcal{T}_j$, $\beta(\boldsymbol{u}) = |\{1 \leq i \leq \ell : \boldsymbol{u}^T \cdot \boldsymbol{h}'_i = \mathbf{0}\}| \geq r - j$. Then since the sets $\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_{r-1}$ are non-overlapping (they have no common elements), we can use Theorem 14 with $\boldsymbol{b} = \mathbf{0}$ to obtain $f(\mathbf{0}) = \frac{1}{3^r} \sum_{\boldsymbol{a} \in \mathcal{A}} 4^{\beta(\boldsymbol{a})} \geq \frac{1}{3^r} |\mathcal{T}_0| 4^\ell + \frac{1}{3^r} \sum_{j=1}^{r-1} |\mathcal{T}_j| 4^{r-j}$. Finally,

$$\begin{aligned}
f(\mathbf{0}) &\geq \frac{1}{3^r} |\mathcal{T}_0| 4^\ell + \frac{1}{3^r} \sum_{j=1}^{r-1} |\mathcal{T}_j| 4^{r-j} \\
&\geq \frac{1}{3^r} 4^\ell + \frac{1}{3^r} \sum_{j=1}^{r-1} (3^j - 3^{j-1}) 4^{r-j} \\
&= \frac{1}{3^r} 4^\ell + \frac{2 \cdot 4^{r-1}}{3^r} \sum_{j=0}^{r-2} \left(\frac{3}{4}\right)^j \\
&= \frac{1}{3^r} 4^\ell + 2 \left(\frac{4}{3}\right)^r - 2 \cdot \frac{4}{3},
\end{aligned}$$

and therefore the proof is complete. □

We summarize the result from Lemma 42 with the following corollary.

Corollary 20. *Let $\mathcal{C}'$ be a t-random-error-correcting ternary code of length $\ell = \frac{n}{2}$ (where n is an even integer) with a parity check matrix H' of dimension r. For $\boldsymbol{a} \in \mathcal{A}$, let $\mathcal{C}_{\boldsymbol{a}}$ be a code created according to Construction G of length n with the constituent code $\mathcal{C}'$. Then for any $\boldsymbol{a} \in \mathcal{A}$, $|\mathcal{C}_{\boldsymbol{a}}| \leq |\mathcal{C}_{\mathbf{0}}|$ and $|\mathcal{C}_{\mathbf{0}}| \geq \lceil \frac{2^n}{3^r} + 2 \left(\frac{4}{3}\right)^r - \frac{8}{3} \rceil$.*

Proof. From Claim 23, $|\mathcal{C}_a| = f(\boldsymbol{a})$. Using Corollary 19, we have that for any $\boldsymbol{a} \in \mathcal{A}$, $|\mathcal{C}_{\boldsymbol{a}}| = f(\boldsymbol{a}) \leq f(\mathbf{0}) = |\mathcal{C}_{\mathbf{0}}|$. Combining Claim 23 and Lemma 42 gives that $|\mathcal{C}_{\mathbf{0}}| = f(\mathbf{0}) \geq \frac{4^\ell}{3^r} + 2 \left(\frac{4}{3}\right)^r - 2 \cdot \frac{4}{3}$. □

Thus, the previous corollary improved upon Corollary 18 where it was shown that for some $\boldsymbol{a} \in \mathcal{A}$, $|\mathcal{C}_{\boldsymbol{a}}| \geq \frac{2^n}{3^r}$. For the case of $t = 2, 3, 4$, we compared our lower bound with the cardinality of the t-grain-error-correcting codes from Table 5.3. Each entry in Table 5.4 contains two numbers delimited by a '/'. The first number is the cardinality of a t-grain-error-correcting code created according to Construction F (from Table 5.3) and the second number is the lower bound from Corollary 20. It can be seen in Table 5.4 that the difference between the bound from Corollary 20 and the size of the codes from Table 5.3 is small for the $t = 2$ case.

In the next section, we return to the problem of constructing single mineral codes.

5.5 General Single-Grain and Single-Mineral Codes from Construction F

In this section, we consider single-mineral codes derived from more general colorings according to Construction F. In Section 5.5.1, we investigate a sufficient condition for codes created with Construction F to produce large single-mineral codes. In Section 5.5.2, we consider the cardinalities of codes created according to Construction F given a coloring based on the group codes [CR79]. In Section 5.5.3, we describe a coloring that was found using a computerized search, and for code lengths 48 and 342 this coloring produces new codes with large cardinalities (larger than using the alternative group codes to construct single mineral-codes).

Recall from Construction F in Section 5.3.2 that the construction for a t-mineral-error-correcting code $\mathcal{C}$ relied on two key ingredients:

1. a mapping $\Phi_{t,m}$ from $GF(2)^m$ to p color classes (where p is a prime), and

2. a t-random-error-correcting code $\mathcal{C}_t$ over $GF(p)$.

The basic idea behind Construction F was to use $\Phi_{t,m}$ to map the color classes of a p-coloring onto the symbols of the non-binary code $\mathcal{C}_t$.

Thus far, we have considered code constructions for mineral codes using Construction F with the map $\Phi_{t,m} = \Gamma$, where Γ is given by (5.7). Therefore, if Construction F is used to create mineral codes, there are two possible directions to investigate:

1. discover new mappings $\Phi_{t,m}$ for $m \geq 2$, and

2. investigate codes for $\mathcal{C}_t$ that, when used in conjunction with some $\Phi_{t,m}$, result in codes with large cardinalities.

In this section, we focus on the first direction for the case where $t = 1$, where the code $\mathcal{C}_1$ is a single random-error-correcting code that is a Hamming code. The second item highlights a potential area of future work which we will discuss briefly in the next section.

In the first subsection, we show that if $\Phi_{t,m}$ has $p = m+1$ color classes where p is a prime, then it is possible to construct single-mineral codes that have at least as many codewords as perfect single random-error-correcting binary codes of the same length. In the second subsection, single-mineral codes created using a coloring scheme based upon the group codes are considered. Motivated by the insights from the first two subsections, we derive new codes of lengths 48 and 342 in the third subsection. These new codes are larger than any codes of the same length produced according to Construction E.

5.5.1 A sufficient condition for Construction F to produce large codes

Suppose that a single-mineral code $\mathcal{C}$ of length mn is created according to Construction F. Suppose that the p-coloring $\Phi_{1,m}$ is such that $p = m + 1$ where p is an odd prime and $\mathcal{C}_1$ is a perfect non-binary single random-error-correcting code over $GF(p)$ of length n. We show that there exists a mineral code $\mathcal{C}$ of length mn whose cardinality is at least $\frac{2^{mn}}{mn+1}$. Motivated by this observation, in Sections 5.5.2 and 5.5.3, we consider using different coloring schemes (i.e., where $\Phi_{1,m} \neq \Gamma$) in conjunction with a perfect single-random-error correcting code. We first begin by reviewing some notation that was used in Section 5.3.2.

As in Section 5.3.2, let $\mathcal{G}_{t,m} = (V, E)$ denote a simple graph where $V = GF(2)^m$, and for any $\boldsymbol{x}, \boldsymbol{y} \in V$ (where $\boldsymbol{x} \neq \boldsymbol{y}$) $(\boldsymbol{x}, \boldsymbol{y}) \in E$ if $\mathcal{B}_{t,M}(\boldsymbol{x}) \cap \mathcal{B}_{t,M}(\boldsymbol{y}) \neq \emptyset$. Recall that $\Phi_{t,m} : GF(2)^m \to GF(p)$ is a p-coloring if it assigns different elements of $GF(p)$ to adjacent vertices. From Section 5.1.3, $\chi(\mathcal{G}_{t,m})$ is the smallest p for which a p-coloring is possible. Recall, the size of the largest clique in a graph $\mathcal{G}$ is denoted $\varsigma(\mathcal{G})$.

The following claim will be used in the proof of Lemma 43.

Claim 24. *For any $m \geq 2$, $\varsigma(\mathcal{G}_{1,m}) \geq m + 1$.*

Proof. Let $\mathcal{S} = \{\boldsymbol{x} \in GF(2)^m : wt(\boldsymbol{x}) \leq 1\}$. Since for any $\boldsymbol{x} \in \mathcal{S}$, $\mathcal{B}_{1,m}(\boldsymbol{x})$ contains the all-zeros vector, it follows that $\mathcal{S}$ is a clique in $\mathcal{G}_{1,m}$. Since $|\mathcal{S}| = m + 1$, the result follows. □

Lemma 43. *Let m be a positive integer. Then, $\chi(\mathcal{G}_{1,m}) = m + 1$.*

Proof. We first show that $\chi(\mathcal{G}_{1,m}) \leq m + 1$. Suppose $\mathcal{A}$ is an Abelian group. Let $a \in \mathcal{A}$ and consider a single-grain code $\mathcal{C}_a^{\mathcal{A}}$ of length $|\mathcal{A}| = m + 1$ created using Construction E. Let $\tilde{\mathcal{C}}_a^{\mathcal{A}}$ be the group code of length m that is the result of shortening the codewords in $\mathcal{C}_a^{\mathcal{A}}$ on the first bit. From Corollary 15, $\tilde{\mathcal{C}}_a^{\mathcal{A}}$ is a single-mineral code. Assign to every $\boldsymbol{x} \in \tilde{\mathcal{C}}_a^{\mathcal{A}}$ the same number from $\{0, 1, \ldots, m\}$. Repeating this process for every value of $a \in \mathcal{A}$ (and using a different number for different values of a), results in an $(m+1)$-coloring on the graph $\chi(\mathcal{G}_{1,m})$ since there are $|\mathcal{A}| = m + 1$ choices for a.

Recall from Section 5.1.3 that $\chi(\mathcal{G}_{1,m}) \geq \varsigma(\mathcal{G}_{1.m})$ where $\varsigma(\mathcal{G}_{1,m})$ is the maximum size of any clique in the graph $\mathcal{G}_{1,m}$. From Claim 24, we have $\chi(\mathcal{G}_{1,m}) \geq \varsigma(\mathcal{G}_{1,m}) \geq m + 1$ and so $\chi(\mathcal{G}_{1,m}) = m + 1$. □

The following theorem is similar to Corollary 18.

Theorem 15. *Let p be a prime number and r a positive integer where $n = \frac{p^r-1}{p-1}$ and $m = p - 1$. Then there exists a single-mineral code $\mathcal{C}$ of length mn where $|\mathcal{C}| \geq \frac{2^{mn}}{mn+1}$ from Construction F.*

Proof. Let $\mathcal{C}_1$ be the constituent non-binary code from Construction F of length n with a parity check matrix H' of dimension r and suppose that $\mathcal{C}_1$ is perfect and $\mathcal{A} = GF(p)^r$. For $\boldsymbol{a} \in \mathcal{A}$, let $\mathcal{C}'_{\boldsymbol{a}} = \{\boldsymbol{x}' \in GF(p)^n : H' \cdot \boldsymbol{x}' = \boldsymbol{a}\}$. Notice that since $\mathcal{C}_1$ is a perfect single random-error-correcting code then $\mathcal{C}'_{\boldsymbol{a}}$ is also a perfect single random-error-correcting code. Thus, we can apply Construction F to obtain a single-mineral code $\mathcal{C}_{\boldsymbol{a}}$ where

$$\mathcal{C}_{\boldsymbol{a}} = \{\boldsymbol{x} \in GF(2)^{mn} : \Phi_{1,m}(\boldsymbol{x}) \in \mathcal{C}'_{\boldsymbol{a}}\}.$$

Since $\Phi_{1,m}$ maps every element in $GF(2)^{mn}$ to exactly one non-binary vector of length n, it follows that every $\boldsymbol{x} \in GF(2)^{mn}$ belongs to exactly one $\mathcal{C}_{\boldsymbol{a}}$, and so the codes $\mathcal{C}_{\boldsymbol{a}_1}, \mathcal{C}_{\boldsymbol{a}_2}, \ldots, \mathcal{C}_{\boldsymbol{a}_{p^r}}$ partition the space $GF(2)^{mn}$ into p^r non-overlapping sets. By the pigeonhole principle, there exists a $\boldsymbol{b} \in \mathcal{A}$, where $|\mathcal{C}_{\boldsymbol{b}}| \geq \frac{2^{mn}}{p^r} = \frac{2^{mn}}{mn+1}$. □

We now consider using the coloring scheme discussed in the proof of Lemma 43 to produce single-mineral codes. More precisely, let $\psi_m : GF(2)^m \to GF(m+1)$ be the mapping so that for a vector $\boldsymbol{x} \in GF(2)^m$,

$$\psi_m(\boldsymbol{x}) = \sum_{i=1}^{m} i x_i \mod m+1.$$

Then, let $A_j = \{\boldsymbol{y} \in GF(2)^m : \psi_m(\boldsymbol{y}) = j\}$. We refer to the vector $(A_0, A_1, \ldots, A_m)$ as the ***group-code partition***. Let Π_m be the set of permutations of the symbols $0, 1, \ldots, m$. For example, the permutation $(1, 0, 2)$ is an element in Π_2. Then, for any permutation $\boldsymbol{a} = (a_0, \ldots, a_m) \in \Pi_m$, we define a coloring $\Phi_{\boldsymbol{a}} : GF(2)^m \to GF(m+1)$ as follows

$$\Phi_{\boldsymbol{a}}(\boldsymbol{x}) = a_{\psi_m(\boldsymbol{x})}.$$

We provide an example illustrating this mapping.

Example 17. *Let* $\boldsymbol{a} = (1, 0, 2)$ *so that* $m = 2$. *Let* $\boldsymbol{x}_1 = (1, 1)$ *so that* $\Phi_{\boldsymbol{a}}(\boldsymbol{x}_1) = a_{\psi_2(\boldsymbol{x}_1)} = a_0 = 1$. *Similarly for* $\boldsymbol{x}_2 = (0, 1)$, *we have* $\Phi_{\boldsymbol{a}}(\boldsymbol{x}_2) = a_{\psi_2(\boldsymbol{x}_2)} = a_2 = 2$.

Suppose the single-mineral code $\mathcal{C}$ is constructed according to Construction F with $\Phi_{\boldsymbol{a}}$

and the single random-error-correcting non-binary code $\mathcal{C}_1$ with symbols over $GF(m+1)$. For shorthand, we refer to $\mathcal{C}$ as $\mathcal{C}(\boldsymbol{a}, \mathcal{C}_1)$.

In the next subsection, we determine which choice of $\boldsymbol{a}$ maximizes the cardinality $|\mathcal{C}(\boldsymbol{a}, \mathcal{C}_1)|$ when $\mathcal{C}_1$ is a linear code. In Section 5.5.3 a different map $\Phi_{1,6}$ is derived using a computerized search over the space $GF(2)^6$ and using this map better single-mineral codes are found for certain code lengths.

5.5.2 Single-mineral codes created using the group-code partition

In this section, we consider the problem of which choice of $\boldsymbol{a} \in \Pi_m$ maximizes $|\mathcal{C}(\boldsymbol{a}, \mathcal{C}_1)|$ when $\mathcal{C}_1$ is a linear code. We show that any $\boldsymbol{a} = (a_0, \ldots, a_m) \in \Pi_m$ where $a_0 = 0$ maximizes the cardinality of the resulting single-mineral code. Since the largest color class under ψ_m is A_0 (cf. [CR79]), one such choice for $\boldsymbol{a}$ is the identity (i.e., $\boldsymbol{a} = (0, 1, 2 \ldots, m)$). For shorthand, let $\boldsymbol{i} = (0, 1, 2, \ldots, m)$. We show that the cardinality of $\mathcal{C}(\boldsymbol{i}, \mathcal{C}_1)$ is exactly the same as the cardinality of a code created according to the Constantin-Rao construction (for codes of the same length) [CR79].

For a non-binary code $\mathcal{C}$ with symbols from $GF(p)$ where p is an odd prime, let $W_{i_0,\ldots,i_{p-1}}$ denote the number of codewords in $\mathcal{C}$ that have exactly i_0 symbols of value 0, i_1 symbols of value 1, and so on. We denote the ***complete weight enumerator*** of $\mathcal{C}$ as

$$W_{\mathcal{C}}(z_0, \ldots, z_{p-1}) = \sum_{i_0, i_1, \ldots, i_{p-1}} W_{i_0,\ldots,i_{p-1}} z_0^{i_0} \cdots z_{p-1}^{i_{p-1}}.$$

We make use of the following known claim [HAL13] [ROT06] in Theorem 16, which will be proven using the MacWilliams Theorem. Recall ζ_p is a p-th root of unity and p is a prime. For an integer i, ζ_p^i denotes the i-th power of ζ_p.

Claim 25. *(c.f. [HAL13],[ROT06]) Suppose $\mathcal{C}$ is a linear code of length n with symbols over $GF(p)$. Then, the complete weight enumerator $W_{\mathcal{C}}(z_0, \ldots, z_{p-1})$ can be written in terms of*

the codewords in the dual code $\mathcal{C}^{\perp}$ (of $\mathcal{C}$) as follows

$$\frac{1}{|\mathcal{C}^{\perp}|} \sum_{\boldsymbol{c}=(c_1,\ldots,c_n)\in\mathcal{C}^{\perp}} \prod_{i=1}^{n} \left(z_0 + z_1\zeta_p^{1\cdot c_i} + \ldots + z_{p-1}\zeta_p^{(p-1)\cdot c_i} \right).$$

Proof. The approach used will be the same as that in ([HAL13], Chapter 9) and ([ROT06], Chapter 4). Recall from Section 5.5.2, we write the complete weight enumerator of a code $\mathcal{C}$ as

$$W_{\mathcal{C}}(z_0,\ldots,z_{p-1}) = \sum_{i_0,i_1,\ldots,i_{p-1}} W_{i_0,\ldots,i_{p-1}} z_0^{i_0}\cdots z_{p-1}^{i_{p-1}}$$

where $W_{i_0,\ldots,i_{p-1}}$ denotes the number of codewords in a code $\mathcal{C}$ that have exactly i_0 symbols of value 0, i_1 symbols of value 1, and so on.

Suppose ζ_p is a p-th root of unity. Then the complete weight enumerator of a code $\mathcal{C}$ of length n defined over $GF(p)$ can be expressed in terms of the codewords in the dual code $\mathcal{C}^{\perp}$ as follows. For shorthand, let $W_{\mathcal{C}}(z_0,\ldots z_{p-1}) = W_{\mathcal{C}}(\boldsymbol{z})$. First, note that for $\boldsymbol{v}\in GF(p)^n$,

$$\sum_{\boldsymbol{x}\in\mathcal{C}^{\perp}} \zeta_p^{\boldsymbol{v}^T\cdot\boldsymbol{x}} = \begin{cases} |\mathcal{C}^{\perp}| & \text{if } \boldsymbol{v}\in\mathcal{C}, \\ 0 & \text{otherwise.} \end{cases} \tag{5.16}$$

Let $I_{\mathcal{C}} : GF(p)^n \to \{0,1\}$ denote the indicator function where for $\boldsymbol{x}\in GF(p)^n$

$$I_{\mathcal{C}}(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x}\in\mathcal{C}, \\ 0 & \text{otherwise.} \end{cases} \tag{5.17}$$

Then,

$$\begin{aligned}
W_{\mathcal{C}}(\boldsymbol{z}) &= \sum_{\boldsymbol{v}\in GF(p)^n} I_{\mathcal{C}}(\boldsymbol{v}) W_{\{\boldsymbol{v}\}}(\boldsymbol{z}) \\
&= \sum_{\boldsymbol{v}\in GF(p)^n} \frac{1}{|\mathcal{C}^\perp|} \left(\sum_{\boldsymbol{x}\in\mathcal{C}^\perp} \zeta_p^{\boldsymbol{v}^T\cdot\boldsymbol{x}} \right) W_{\{\boldsymbol{v}\}}(\boldsymbol{z}) \\
&= \frac{1}{|\mathcal{C}^\perp|} \sum_{\boldsymbol{x}\in\mathcal{C}^\perp} \sum_{\boldsymbol{v}\in GF(p)^n} \zeta_p^{\boldsymbol{v}^T\cdot\boldsymbol{x}} W_{\{\boldsymbol{v}\}}(\boldsymbol{z}) \\
&= \frac{1}{|\mathcal{C}^\perp|} \sum_{\boldsymbol{x}\in\mathcal{C}^\perp} \sum_{\boldsymbol{v}\in GF(p)^n} \prod_{i=1}^{n} \zeta_p^{v_i x_i} W_{\{v_i\}}(\boldsymbol{z}) \\
&= \frac{1}{|\mathcal{C}^\perp|} \sum_{\boldsymbol{x}\in\mathcal{C}^\perp} \prod_{i=1}^{n} \sum_{v_i\in GF(p)} \zeta_p^{v_i x_i} W_{\{v_i\}}(\boldsymbol{z}) \\
&= \frac{1}{|\mathcal{C}^\perp|} \sum_{\boldsymbol{x}\in\mathcal{C}^\perp} \prod_{i=1}^{n} \left(z_0 + \zeta_p^{1\cdot x_i} z_1 + \ldots + \zeta_p^{(p-1)\cdot x_i} z_{p-1} \right).
\end{aligned}$$

□

The next claim will also be useful in the proof of Theorem 16.

Claim 26. *Let j^*, c be integers such that $0 \leq j^* \leq p-1$ and $c \neq 0(\mod p)$. Then, $\sum_{j=0, j\neq j^*}^{p-1} \zeta_p^{j\cdot c} = -\zeta_p^{j^*\cdot c}$.*

We make use of the following notation in the statement of the next claim. For any $\boldsymbol{a} \in \Pi_m$ (recall Π_m is the set of permutations of the symbols $0, 1, \ldots, m$) and any integer k where $0 \leq k \leq m$ let $\boldsymbol{a}(k)$ denote the index in $\boldsymbol{a}$ of the number k.

Claim 27. *Let $m+1$ be an odd prime. Suppose $W_{\mathcal{C}_1}(z_0, \ldots, z_m)$ is the complete weight enumerator for a non-binary $(m+1)$-ary code $\mathcal{C}_1$. Then for any $\boldsymbol{a} \in \Pi_m$, $|\mathcal{C}(\boldsymbol{a}, \mathcal{C}_1)| = W_{\mathcal{C}_1}(|A_{\boldsymbol{a}(0)}|, \ldots, |A_{\boldsymbol{a}(m)}|)$.*

We are now ready to state the main result of this subsection.

Theorem 16. *Suppose $\mathcal{C}_1$ is a linear code over $GF(m+1)$ where $m+1$ is an odd prime. For any $\boldsymbol{a} \in \Pi_m$, $|\mathcal{C}(\boldsymbol{a}, \mathcal{C}_1)|$ is maximized when $a_0 = 0$. Furthermore for any $\boldsymbol{b} \in \Pi_m$ where $b_0 = 0$, $|\mathcal{C}(\boldsymbol{a}, \mathcal{C}_1)| = |\mathcal{C}(\boldsymbol{b}, \mathcal{C}_1)|$.*

Proof. Let $p = m + 1$ and $\boldsymbol{a} = (a_0, \ldots, a_m)$. Under the group-code partition, the largest color class A_0 has cardinality $\frac{2^{p-1}+p-1}{p}$ and the other color classes have cardinality $\frac{2^{p-1}-1}{p}$ ([CR79]). We prove the theorem by considering the cardinality of the code created according to Construction F when the color classes from the group-code partition are mapped to different symbols in $GF(p)$. From Claim 27, the cardinality of the mineral code $\mathcal{C}(\boldsymbol{a}, \mathcal{C}_1)$ created according to Construction F can be derived from $W_{\mathcal{C}_1}(z_0, \ldots, z_{p-1})$ by substituting for each z_i, the size of the color class that is mapped to symbol i (as a result of the permutation $\boldsymbol{a}$). Suppose $\mathcal{C}_1^{\perp}$ represents the dual code of $\mathcal{C}_1$. Then, from Claims 25 and 27, we can write $|\mathcal{C}(\boldsymbol{a}, \mathcal{C}_1)| = \frac{1}{|\mathcal{C}_1^{\perp}|} \sum_{\boldsymbol{c} \in \mathcal{C}_1^{\perp}} \prod_{i=1}^{n} (|A_{\boldsymbol{a}(0)}| + |A_{\boldsymbol{a}(1)}| \zeta_p^{1 \cdot c_i} + \ldots + |A_{\boldsymbol{a}(p-1)}| \zeta_p^{(p-1) \cdot c_i})$.

In particular, we consider the term

$$\Lambda(\boldsymbol{a}, c_i) = \left(|A_{\boldsymbol{a}(0)}| + |A_{\boldsymbol{a}(1)}| \zeta_p^{1 \cdot c_i} + \ldots + |A_{\boldsymbol{a}(p)}| \zeta_p^{(p-1) \cdot c_i} \right) \tag{5.18}$$

for certain choices of $\boldsymbol{a}$ and $c_i \in GF(p)$. Note that under this setup $|\mathcal{C}(\boldsymbol{a}, \mathcal{C}_1)| = \frac{1}{|\mathcal{C}_1^{\perp}|} \sum_{\boldsymbol{c} \in \mathcal{C}_1^{\perp}} \prod_{i=1}^{n} \Lambda(\boldsymbol{a}, c_i)$.

We consider two cases:

1. $a_0 = 0$, or
2. $a_0 \neq 0$.

In the remainder of the proof we refer to the setup in item 1) above as Case 1) and the setup in item 2) above as Case 2). Notice that under either Case 1) or Case 2), we have $\Lambda(\boldsymbol{a}, 0) = \sum_{k=0}^{p-1} |A_{\boldsymbol{a}(k)}| = 2^{p-1}$. In the following two cases, we therefore only consider the quantity $\Lambda(\boldsymbol{a}, c)$ where $c \in GF(p)$ and $c \neq 0$.

Case 1: Suppose $\boldsymbol{a}$ is such that $a_0 = 0$. Then $\Lambda(\boldsymbol{a}, c_i)$ (where $c_i \neq 0, c_i \in GF(p)$) can be written as

$$\Lambda(\boldsymbol{a}, c_i) = \frac{2^{p-1} + p - 1}{p} + \frac{2^{p-1} - 1}{p} \sum_{k=1}^{p-1} \zeta_p^{k \cdot c_i}.$$

Applying Claim 26, we get that $\Lambda(\boldsymbol{a}, c_i) = 1$ when $c_i \neq 0$.

Case 2: Suppose $\boldsymbol{a}$ is such that $a_0 \neq 0$. In particular, we assume $a_{j^*} = 0$ for $j^* \neq 0$.

Then, we can write $\Lambda(\boldsymbol{a}, c_i)$ as (where $c_i \neq 0, c_i \in GF(p)$)

$$\begin{aligned}\Lambda(\boldsymbol{a}, c_i) &= \frac{2^{p-1}+p-1}{p}\zeta_p^{j^* c_i} + \frac{2^{p-1}-1}{p}\sum_{j=0, j\neq j^*}^{p-1}\zeta_p^{jc_i} \\ &= \frac{2^{p-1}+p-1}{p}\zeta_p^{j^* c_i} + \frac{2^{p-1}-1}{p}\left(-\zeta_p^{j^* c_i}\right) \\ &= \zeta_p^{j^* c_i}.\end{aligned}$$

Notice that $|\zeta_p^{j^* c_i}| \leq 1$ for any integer $j^* \neq 0$.

Summary: Using the ideas from above, we now show that $\mathcal{C}(\boldsymbol{a}, \mathcal{C}_1)$ is maximized when $a_0 = 0$. Consider any $\boldsymbol{b} = (b_0, b_1, \ldots, b_m), \boldsymbol{d} = (d_0, d_1, \ldots, d_m) \in \Pi_m$ where $b_0 = 0 \neq d_0$. From the previous analysis, for any $c_i \in GF(p)$ we have $|\Lambda(\boldsymbol{d}, c_i)| \leq \Lambda(\boldsymbol{b}, c_i)$ and so

$$\frac{1}{|\mathcal{C}_1^{\perp}|}\sum_{\boldsymbol{c}\in\mathcal{C}_1^{\perp}}\prod_{i=1}^{n}\Lambda(\boldsymbol{d}, c_i) \leq \frac{1}{|\mathcal{C}_1^{\perp}|}\sum_{\boldsymbol{c}\in\mathcal{C}_1^{\perp}}\prod_{i=1}^{n}\Lambda(\boldsymbol{b}, c_i),$$

and

$$|\mathcal{C}(\boldsymbol{d}, \mathcal{C}_1)| \leq |\mathcal{C}(\boldsymbol{b}, \mathcal{C}_1)|.$$

Let $\boldsymbol{g} = (g_0, g_1, \ldots, g_m) \in \Pi_m$ where $g_0 = 0$ but $\boldsymbol{g} \neq \boldsymbol{b}$. We have left to show that for any such $\boldsymbol{g}$, $|\mathcal{C}(\boldsymbol{g}, \mathcal{C}_1)| = |\mathcal{C}(\boldsymbol{b}, \mathcal{C}_1)|$ where $\boldsymbol{b}$ is as defined in the previous paragraph. From the previous analysis, for any $c_i \in GF(p)$ we have $\Lambda(\boldsymbol{d}, c_i) = \Lambda(\boldsymbol{b}, c_i)$ and so $|\mathcal{C}(\boldsymbol{d}, \mathcal{C}_1)| = \frac{1}{|\mathcal{C}_1^{\perp}|}\sum_{\boldsymbol{c}\in\mathcal{C}_1^{\perp}}\prod_{i=1}^{n}\Lambda(\boldsymbol{d}, c_i) = \frac{1}{|\mathcal{C}_1^{\perp}|}\sum_{\boldsymbol{c}\in\mathcal{C}_1^{\perp}}\prod_{i=1}^{n}\Lambda(\boldsymbol{b}, c_i) = |\mathcal{C}(\boldsymbol{b}, \mathcal{C}_1)|$. □

From Theorem 16, to maximize the size of a mineral code $\mathcal{C}$ created according to Construction F with a group-code partition, the largest color class A_0 should be mapped to the symbol zero in the constituent code $\mathcal{C}_1$. Suppose the Hamming weight enumerator of a code $\mathcal{C}$ can be written as $W_{\mathcal{C}}(x, z) = \sum_{i=0}^{n} W_{i,n-i} z^i x^{n-i}$ where $W_{i,n-i}$ represents the number of codewords in $\mathcal{C}$ whose Hamming weight is i. The following result follows from Theorem 16.

Corollary 21. *Let $m+1$ be an odd prime integer. Let $\mathcal{C}$ be a single-mineral code created according to Construction F where the group-code partition is used and the $(m+1)$-ary constituent code $\mathcal{C}_1$ is a non-binary Hamming code of length n. Then $|\mathcal{C}| \leq \frac{2^{mn}}{mn+1} + \frac{mn2^{(m(n-1))/(m+1)}}{mn+1}$.*

Proof. For the non-binary Hamming code $\mathcal{C}_1$ of length n defined over $GF(m+1)$, we have that

$$W_{\mathcal{C}_1}(x,z) = \frac{1}{mn+1}\left((x+mz)^n + mn(x-z)^{(mn+1)/(m+1)}(x+mz)^{(n-1)/(m+1)}\right)$$

from ([ROT06], Chapter 4). Recall that under the group-code partition, the largest color class A_0 has cardinality $\frac{2^m+m}{m+1}$ and the other color classes have cardinality $\frac{2^m-1}{m+1}$ ([**?**]). In other words, we have $|A_{\boldsymbol{a}(1)}| = |A_{\boldsymbol{a}(2)}| = \cdots = |A_{\boldsymbol{a}(p-1)}|$. Furthermore from Theorem 16, we have that $|\mathcal{C}(\boldsymbol{a}, \mathcal{C}_1)|$ is maximized when $a_0 = 0$. Thus, substituting $x = \frac{2^m+m}{m+1}$ and $z = \frac{2^m-1}{m+1}$ then gives the maximum number of codewords in the code $\mathcal{C}$ according to Theorem 16. □

In the following remark, recall Π_m refers to the set of permutations of the symbols $\{0, 1, \ldots, m\}$.

Remark 10. *For a prime p, a positive integer $r \geq 2$, and the Abelian group $\mathcal{A} = GF(p)^r$, it was shown in [CR79], Theorem 14, that the length $p^r - 1$ code $\mathcal{C}_0^{\mathcal{A}}$ satisfies $|\mathcal{C}_0^{\mathcal{A}}| = \frac{2^{p^r-1}}{p^r} + \frac{(p^r-1)2^{p^{r-1}-1}}{p^r}$. Let $\boldsymbol{a} = (0, 1, \ldots, p-1) \in \Pi_{p-1}$ and suppose $\mathcal{C}_1$ is a perfect code of length $\frac{p^r-1}{p-1}$ over $GF(p)$ so that $\mathcal{C}(\boldsymbol{a}, \mathcal{C}_1)$ has length $p^r - 1$. Then from Corollary 21, $|\mathcal{C}(\boldsymbol{a}, \mathcal{C}_1)| = |\mathcal{C}_0^{\mathcal{A}}|$.*

As noted in the previous remark, if Construction F is used to create a single-mineral code and $\mathcal{C}_1$ is a perfect and linear non-binary code, then (for a fixed length) Construction F does not result in codes that are any larger than the group codes. In the next section, we consider using perfect and linear non-binary single-random-error-correcting codes with different coloring schemes to construct larger codes.

5.5.3 A new coloring scheme

In this section, we report on the results of using Construction F with a new map that was located using a computerized search. As before, we denote the color classes as A_0, A_1, ..., A_{k-1} for the k-coloring $\Phi_{t,m}$ on $\mathcal{G}_{t,m}$ where $A_0 \cup A_1 \cup \cdots \cup A_{p-1} = GF(2)^m$. By this setup, we assume

1. $\forall j \in \{0, \ldots, k-1\}$, $A_j \subseteq GF(2)^m$,
2. for any $i, j \in \{0, \ldots, k-1\}$ where $i \neq j$ $A_i \cap A_j = \emptyset$,
3. $|A_0| \geq |A_1| \geq \ldots \geq |A_{k-1}|$.

In this subsection, we make use of the following notation. Suppose a code $\mathcal{C}$ is a t-mineral-error-correcting code created according to Construction F given by

1. a set of p color classes $D = \{A_0, A_1, \ldots, A_{p-1}\}$ for a p-coloring on $\mathcal{G}_{t,m}$ where p is a prime,
2. the mapping $\Phi_{t,m}$ which maps vectors from $GF(2)^m$ into the symbols $\{0, 1, \ldots, p-1\}$,
3. $\mathcal{C}_t$ where $\mathcal{C}_t$ is a t-random-error-correcting code over $GF(p)$.

We denote the mineral code $\mathcal{C}$ as $\mathcal{C}(\Phi_{t,m}, \mathcal{C}_t)$. Under this setup, the map $\Phi_{t,m}$ always maps elements from the same color class to the same symbol.

In the following, we describe the color classes from a 7-coloring on $\mathcal{G}_{1,6}$ that was located with the aid of a computer search. The vectors from $GF(2)^m$ are enumerated by their decimal representation. For example, the vector $\boldsymbol{x} = (x_1, x_2, x_3, x_4, x_5, x_6) = (1, 0, 1, 1, 0, 0)$ corresponds to the number 13 since $\sum_{i=1}^{6} 2^{i-1} x_i = 13$ in this representation. The color classes are the following:

Color class A_0

$\{0, 3, 12, 15, 21, 24, 36, 43, 49, 54, 61\}$

Color class A_1

$\{1, 6, 13, 18, 25, 30, 37, 40, 59\}$

Color class A_2

$\{1, 6, 13, 18, 25, 30, 37, 40, 59\}$

Color class A_3

$\{4, 7, 9, 19, 31, 34, 46, 52, 57\}$

Color class A_4

$\{8, 11, 20, 23, 33, 38, 45, 50, 62\}$

Color class A_5

$\{10, 16, 22, 28, 35, 41, 47, 53, 58\}$

Color class A_6

$\{2, 5, 14, 27, 42, 48, 55, 60\}$.

Notice that $|A_0| = 11, |A_1| = 9, |A_2| = 9, |A_3| = 9, |A_4| = 9, |A_5| = 9$, and $|A_6| = 8$. Recall that if the group-code partition was used then the sizes of the color classes are $10, 9, 9, 9, 9, 9, 9$ so that the size of the largest color class has increased by 1 given the new set of color classes.

Using a non-binary perfect code over $GF(7)$ of length 8 with the coloring scheme mentioned in this section, the resulting length-48 binary code has 16192 more codewords than a

group code defined over $\mathbb{Z}_7 \times \mathbb{Z}_7$. Using a non-binary perfect code over $GF(7)$ of length 57 with the coloring scheme described in this section results in a binary code of length 342 with approximately $7.1401e34$ more codewords than a group code defined over $\mathbb{Z}_7 \times \mathbb{Z}_7 \times \mathbb{Z}_7$. The parity check matrices for the single-random-error-correcting codes of length 8 and of length 57 over $GF(7)$ were taken from [MAG11].

5.6 Conclusion

Recall from Chapter 1, that when an error occurs within a granular media recording medium, the errors largely manifest themselves as smears whereby the information in one bit overwrites the information in another bit. In this chapter, we presented bounds and constructions for codes of this type that improve upon the existing results. Since the codes presented here have higher rates than traditional error-correcting codes (while correcting the same number of grain-errors), the proposed codes offer an attractive alternative to traditional error-correcting codes in the context of granular recording media devices.

CHAPTER 6

Conclusion

6.1 Summary of Our Results

Despite the promise of greater storage densities and faster access times, many types of emerging storage systems exhibit high bit error rates and have shorter lifetimes than traditional storage devices. In this thesis, we explored coding techniques to overcome many of the physical limitations of these storage systems. In particular, this thesis considered four coding problems that had applications to Flash memory and granular media recording. For many problem domains, our codes improved upon existing results.

The first coding problem considered in this thesis was to construct non-binary WOM-codes for multilevel Flash memories. The purpose of WOM-codes is to reduce the number of times a Flash device is erased. Recall from Chapter 1, that since the lifetime of a Flash cell is largely a function of the number of times a cell is erased, WOM-codes have the potential to extend the lifetime of Flash memory. Motivated by this potential benefit, we proposed new codes that relied on mappings between binary and non-binary alphabets. For many cases of interest, the constructions presented in this work produce the best known codes. In addition, the capacity of fixed-rate non-binary codes was considered and expressions for the capacity of t-write WOM-codes were provided.

The second coding problem investigated was to design error-correcting codes for Flash memory. We proposed new codes that correct errors where each cell error only affects a single

bit of information (of the three possible bits). The proposed codes are effective in the context of Flash because when a cell error occurs as a result of the programming procedure, typically the error only affects a single bit of information as described in Chapter 1. The proposed codes were analytically and empirically shown to offer a potentially valuable component for future coding schemes in the context of Flash memory.

The third coding problem we considered was to design codes capable of correcting synchronization errors in a rank modulation system. Recall from Chapter 1 that rank modulation systems can potentially improve the reliability of NVMs by representing information using the relative charge levels of cells rather than absolute values. The goal in this part of the thesis was to further improve the reliability of rank modulation systems by proposing new codes that can correct synchronization errors. It was shown that codes in the Ulam metric are suitable for these types of errors and new codes were proposed. For the case of the single unstable deletion model, a new code code construction was presented and shown to be asymptotically optimal.

The fourth coding problem studied in this thesis was the design of error-correcting codes for granular media recording. Recall from Chapter 1, that errors in granular media recording typically manifest themselves as smears whereby the information from one bit of information smears the information stored in the adjacent bit. The purpose of this chapter was to study and design codes specifically for this type of error model. In particular, new bounds and constructions were derived for grain-error-correcting codes where the lengths of the grains were at most two. We considered a new approach to constructing codes that correct grain-errors and using this approach, we improved upon the constructions in [MBK11] and [SR11].

The preliminary results from this work have been published at ***IEEE International Symposium on Information Theory*** (ISIT) [GD11], [GYGSD12], and [GYD13a] and in ***IEEE International Theory Workshop*** (ITW) [GYDSVW11]. The results from Chapter 4 were submitted to ISIT 2014 [GYFB14a] and [GYFB14b]. The material from Chapter 3 was published in ***IEEE Transactions on Information Theory*** (IT) [GYD12]. The ma-

terial from Chapters 2 and 5 has been submitted to IT [GYD13b], [GD12].

6.2 Future Directions

There are several possibilities available for future work. With regards to coding for Flash memory, an extension of the work in this thesis would be the construction of new WOM-codes that can also correct asymmetric errors. There has been some work done in this area as in [YSVW12], but the model has not been extended to consider any type of asymmetric errors.

There are many potential extensions to our work in error-correcting codes for Flash. For example, the codes presented in Chapter 3 only exploit small number of the asymmetries that are displayed in Table 3.1. A potential area of research would to the design of error-correcting codes for Flash that take into account that errors are more common, for instance, when certain voltages are programmed.

The work presented in Chapter 4 represents work on a new class of synchronization errors. We are currently investigating constructions for codes capable of correcting more than a single unstable insertion and more than a single unstable deletion. A common error that can occur in Flash memory is when the rankings of two cells are swapped. This type of error manifests itself as an adjacent transposition and, motivated by such a setup, codes for rank modulation in the Kendall Tau distance have been proposed [BM10]. A potential area of future research might be on codes that can recover from a prescribed number of adjacent transpositions and unstable deletions.

With regarding to coding for granular media recording, there are also many directions for future work. This includes the development of new coloring schemes and codes to use with Construction F as well as constructions of codes that correct multiple non-overlapping grain-errors. The largest single-grain codes for $9 \leq n \leq 15$ listed in Table 5.2 were the result of using Construction F with non-linear codes over $GF(3)$. It seems promising that potentially larger single-grain codes may be possible using non-linear codes and coloring

schemes in conjunction with Construction F for longer code lengths.

Finally, we note that Construction F may be applicable to the construction of new asymmetric error-correcting codes for the Z-channel. In fact, when $\Phi_{t,m} = \Gamma$ and $\mathcal{C}_1$ (from Construction F) is a single random-error-correcting ternary code, Construction F is identical to the single asymmetric error-correcting code (from the ternary construction) described in [GSSSZ12]. Given new colorings (i.e., where $\Phi_{1,m} \neq \Gamma$) and new ternary codes for $\mathcal{C}_1$, it may be possible to construct new codes with large cardinalities for the Z-channel.

References

[AM10] H. Alhussien and J. Moon, "An iteratively decodable tensor product code with application to data storage," *IEEE J. on Sel. Areas in Comm.*, vol. 28, no. 2, pp. 228 – 240, Feb. 2010.

[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.

[BSU72] W.A. Beyer, M.L. Stein, and S.M. Ulam, "Metric in Biology, an Introduction," Preprint LA-4973, Univ. of Calif., Los Alamos, 1972.

[BAR12] N. Bitouze, A. Graell i Amat, and E. Rosnes, "Making WOM codes decodable using short synchronous WOM codes," *IEEE International Symposium on Information Theory*, Boston, MA, Jul. 2012.

[BV04] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2004.

[BS12] D. Burshtein and A. Strugatski, "Polar write once memory codes," in *IEEE International Symposium on Information Theory*, Boston, MA, Jul. 2012.

[SE13] S. Buzaglo and T. Etzion, "Tilings with n-dimensional chairs and their applications to asymmetric codes," *IEEE Transactions on Information Theory,* vol. 59, no. 3, Feb. 2013, pp. 1573 – 1582.

[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, Apr. 2010, pp. 1582 – 1595.

[CY12] Y. Cassuto and E. Yaakobi, "Short q-ary fixed-rate WOM codes for guaranteed re-writes and with hot/cold write differentiation," submitted to *IEEE Transactions on Information Theory*, 2012.

[CS06a] P. Chaichanavong and P. H. Siegel, "A tensor-product parity code for magnetic recording," *IEEE Transactions on Magnetics*, vol. 42, no. 2, pp. 350-352, Feb. 2006.

[CS06b] P. Chaichanavong and P. H. Siegel, "Tensor-product parity codes: combination with constrained codes and application to perpendicular recording," *IEEE Transactions on Magnetics*, vol. 42, no. 2, pp. 214-219, Feb. 2006.

[CGM86] G. D. Cohen, P. Godlewski, and F. Merkx, "Linear binary code for write-once memories," *IEEE Transactions on Information Theory*, vol. 32, no. 5, Oct. 1986, pp. 697–700.

[CR79] S. D. Constantin and T. R. M. Rao, "On the theory of binary asymmetric error-correcting codes," *Information and Control*, vol. 40, pp. 20-36, 1979.

[COV73] T. Cover, "Enumerative source coding," *IEEE Transactions on Information Theory*, vol. 19, no. 1, Jan. 1973, pp. 73–77.

[CKK12] D. Cullina, A. A. Kulkarni, and N. Kiyavash, "A coloring approach to constructing deletion correcting codes from constant weight subgraphs," *IEEE International Symposium on Information Theory*, Boston, MA, 2012.

[DH98] M. Deza and T. Huang, "Metrics on permutations, a survey," *J. Combin. Inf. Syst. Sci.*, vol. 23, pp. 173–185, 1998.

[DOL10] L. Dolecek, "Towards longer lifetime of emerging memory technologies using number theory," *IEEE Global Communications Conference*, Miami, FL, Dec. 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. 979-983, Mar. 2010.

[FSM13] F. Farnoud (Hassanzadeh), V. Skachek, and O. Milenkovic, "Error-correction in Flash memories via codes in the Ulam metric," *IEEE Transactions on Information Theory*, vol. 59, no. 5, pp. 3003–3020, May 2013.

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

[FLM08] H. Finucane, Z. Liu, and M. Mitzenmacher, "Designing floating codes for expected performance," *46-th Allerton Conference on Communication, Control and Computing*, Monticello, IL, Sep. 2008.

[FV99] F. Fu and A. J. Han Vinck, "On the capacity of generalized write-once memory with state transitions described by an arbitrary directed acyclic graph," *IEEE Transactions on Information Theory*, vol. 45, no. 1, Sep. 1999, pp. 308 – 313.

[GD11] R. Gabrys and L. Dolecek, "Characterizing capacity achieving write once memory codes for multilevel Flash memories," *IEEE International Symposium on Information Theory*, St. Petersburg, Russia, Aug. 2011.

[GYDSVW11] R. Gabrys, E. Yaakobi, L. Dolecek, P. H. Siegel, A. Vardy, and J. K. Wolf, "Non-binary WOM-codes for multilevel Flash memories," *IEEE Information Theory Workshop*, Paraty, Brazil, Oct. 2011.

[GYGSD12] R. Gabrys, E. Yaakobi, L. Grupp, S. Swanson, and L. Dolecek, "Tackling intra-cell variability in TLC Flash through tensor product codes," *IEEE International Symposium on Information Theory*, Boston, MA, Jul. 2012.

[GYD12] R. Gabrys, E. Yaakobi, and L. Dolecek, "Graded-bit-error-correcting codes with applications to Flash memory," *IEEE Transactions on Information Theory*, vol. 59, no. 4, Mar. 2013, pp. 2315-2327.

[GD12] R. Gabrys and L. Dolecek, "Constructions of non-binary WOM-codes for multi-level Flash memories," submitted to *IEEE Transactions on Information Theory*, 2012.

[GYD13a] R. Gabrys, E. Yaakobi, and L. Dolecek, "Correcting grain-errors in magnetic media," *IEEE International Symposium on Information Theory*, Istanbul, Turkey, 2013.

[GYD13b] R. Gabrys, E. Yaakobi, and L. Dolecek, "Correcting grain-errors in magnetic media," submitted to *IEEE Transactions on Information Theory*, 2013.

[GYFB14a] R. Gabrys, E. Yaakobi, F. Farnoud, and J. Bruck, "Codes correcting erasures and deletions for rank modulation," submitted to *IEEE International Symposium on Information Theory*, Honolulu, HI, Jul. 2014.

[GYFB14b] R. Gabrys, E. Yaakobi, F. Farnoud, F. Sala, J. Bruck, and L. Dolecek, "Single-deletion-correcting codes over permutations," submitted to *IEEE International Symposium on Information Theory*, Honolulu, HI, Jul. 2014.

[GSSSZ12] M. Grassl, P. Shor, G. Smith, J. Smolin, and B. Zeng, "New constructions of codes for asymmetric channels via concatenation," *IEEE International Symposium on Information Theory*, Boston, MA, 2012.

[GT05] E. Gal and S. Toledo, "Algorithms and data structures for Flash memories," *ACM Computing Surveys*, vol. 37, no. 2, Jun. 2005, pp. 138–163.

[GOD87] P. Godlewski, "WOM-codes construits à partir des codes de Hamming," *Discrete Math.*, vol. 65, no. 3, Jul. 1987, pp. 237–243.

[GCCSYSW09] L. M. Grupp, A. M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. Siegel, and J. K. Wolf, "Characterizing Flash memory: anomalies, observations, and applications," *42nd Annual IEEE/ACM International Symposium on Microarchitecture*, New York, NY, Dec. 2009.

[GCCSYSW] L. M. Grupp, A. M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. H. Siegel, and J. K. Wolf, "Characterizing Flash memory: anomalies, observations, and applications," *42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009.

[HAL13] J. I. Hall, *Notes on Coding Theory*, available at http://www.mth.msu.edu/∼jhall/classes/codenotes/coding-notes.html, 2013.

[HEE85] C. Heegard, "On the capacity of permanent memory," *IEEE Transactions on Information Theory*, vol. 31, no. 1, Jan. 1985, pp. 34–42.

[HIT14] Hitachi Global Storage Systems, "Patterned Magnetic Media," *https://www1.hgst.com/hdd/research/storage/pm/*, 2014.

[HLA11] Q. Huang, S. Lin, and K. A. S. Abdel-Ghaffar, "Error-correcting codes for Flash coding," *IEEE Transactions on Information Theory*, vol. 57, no. 9, Apr. 2011, pp. 6097 – 6108.

[IF81] H. Imai and H. Fujiya, "Generalized tensor product codes," *IEEE Transactions on Information Theory*, vol. 27, no. 2, pp. 181-187, Mar. 1981.

[ILL89] R. Impagliazzo, L. A. Levin, and M. Luby. "Pseudo-random generation from one- way functions (extended abstracts)," *21st Annual Symposium on Theory of Computing*, Seattle, WA, May 1989.

[ISW11] A. R. Iyengar, P. H. Siegel, and J. K. Wolf, "Write channel model for bit-patterned media recording," *IEEE Transactions on Magnetics*, vol. 47, no. 1, pp. 35-45, 2011.

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

[JMSB09] A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck, "Rank modulation for Flash memories," *IEEE Transations on Information Theory*, vol. 55, no. 6, pp. 2659–2673, Jun. 2009.

[JB09] A. Jiang and J. Bruck, "Information representation and coding for Flash memories," *IEEE Pacific RIM Conference on Communications, Computers, and Signal Processing*, Victoria, B.C., Canada, Aug. 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. 2112–2120, May 2010.

[KG90] M. Kendall and J.D. Gibbons, *Rank Correlation Methods*. New York: Oxford Univ. Press, 1990.

[KK12] A. A. Kulkarni and N. Kiyavash, "Non-asymptotic upper bounds for deletion correcting codes," available at *http://arxiv.org/abs/1211.3128*, 2012.

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

[KIN12] Kingston Technology, "Flash memory guide," available at *http://media.kingston.com/pdfs/FlashMemGuide.pdf.*

[KZ13] N. Kashyap and G. Zemor, "Upper bounds on the size of grain-correcting codes," available at *http://arxiv.org/abs/1302.6154*, 2013.

[KUR11] B. Kurkoski, "Notes on a lattice-based WOM construction that guarantees two-writes," in *34th Symposium on Information Theory and Its Applications*, Ousyuku, Iwate, Japan, Nov.-Dec. 2011.

[LEV66] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet physics doklady*, vol. 10, p. 707–710, 1966.

[LEV91] V. I. Levenshtein, "On perfect codes in deletion and insertion metric," *Discretnaya Mathematika*, vol. 3, no. 1, pp. 3–20, 1991.

[Li08] Y. Li *et al.* "A 16gb 3b/cell nand Flash memory in 56nm with 8mb/s write rate," *IEEE Symposium on Computers and Communications*, Marrakech, Morrocco, Feb. 2008.

[MK09] Y. Maeda and H. Kaneko, "Error control coding for multilevel cell Flash memories using nonbinary low-density parity-check codes," *IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, Chicago, IL, Oct. 2009.

[MAG11] University of Sydney, "Magma Computational Algebra System," *http://magma.maths.usyd.edu.au/magma/*, 2011.

[MBK11] A. Mazumdar, A. Barg, and N. Kashyap, "Coding for high-density recording on a 1-d granular magnetic medium," *IEEE Transactions on Information Theory*, vol. 57, no. 11, pp. 7403-7417, 2011.

[MR79] R. J. McEliece and E. R. Rodemich, "The Constantin-Rao construction for binary asymmetric error-correcting codes," *The Deep Space Network*, pp. 124-129, ID 19790016933, 1979.

[OL08] A. R. Olson and D. J. Langlois, "Solid state drives data reliability and lifetime," *Imation White Paper*, Apr. 2008.

[PB90] S. Park and B. Bose, "Burst asymmetric/unidirectional error correcting/detecting codes," *20th International Symposium on Fault-Tolerant Computing*, pp. 273-280, Newcastle Upon Tyne, UK, 1990.

[ROS04] K.H. Rosen, *Discrete Mathematics and its applications*, Chapman and Hall, 2004.

[ROT06] R. Roth, *Introduction to Coding Theory*, Cambridge University Press, 2006.

[ROT94] J. J. Rotman, *An Introduction to the Theory of Groups*, Springer, 1994.

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

[SS14] W. Ch. Schmid and R. Schürer, "MinT, the online database for optimal parameters of (t, m, s)-nets, (t, s)-sequences, orthogonal arrays, linear codes and OOAs", University of Salzburg, *http://mint.sbg.ac.at/index.php*.

[SCH11] M. Schwartz, "Quasi-cross lattice tilings with applications to Flash memory," *IEEE International Symposium on Information Theory*, St. Petersburgh, Russia, August 2011.

[SR11] A. Sharov and R. M. Roth, "Bounds and constructions for granular media coding," *IEEE International Symposium on Information Theory*, St. Petersburg, Russia, 2011.

[SHI08] N. Shibata *et al.*, "A 70 nm 16 gb 16-level-cell NAND Flash memory," *IEEE d-State Circuits*, vol. 43, no. 4, Apr. 2008, pp. 929–937.

[SHP12a] A. Shpilka, "New constructions of WOM codes using the Wozencraft ensemble," *Latin American Symposium on Theoretical Informatics*, Arequipa, Peru, Apr. 2012.

[SHP12b] A. Shpilka, "Capacity achieving multiwrite WOM codes," *arXiv:1209.1128, http://arxiv.org/abs/1209.1128*, Sep. 2012.

[SDRZ06] F. Sun, S. Devarajan, K. Rose, and T. Zhang, "Multilevel Flash memory on-chip error correction based on trellis coded modulation," *IEEE International Symposium on Circuits and Systems*, Island of Kos, Greece, Sep. 2006.

[TEN84] G. Tenengolts, "Nonbinary codes, correcting single deletion or insertion (Corresp.)," *IEEE Transactions on Information Theory*, vol. 30, no. 5, pp. 766–769, Sep. 1984.

[VT65] R.R. Varshamov and G.M. Tenengolts, "Codes which correct single asymmetric errors (in Russian)," *Avtomatika i Telemekhanika*, vol. 6, no. 2, pp. 288–292, 1965 (trans: *Automation and Remote Contr.*, vol. 26, pp. 286–290).

[WSW11] J. Wang, H. Shankar and R. D. Wesel, "Soft information for LDPC decoding in Flash:mutual-information optimized quantization," *IEEE Global Communications Conference*, Houston, TX, Dec. 2011.

[WES01] D. B. West, *Introduction to Graph Theory*, Prentice Hall Upper Saddle River, 2001, vol. 2.

[WNP97] R. I. White, R. M. H. New and R. F. W. Pease, "Patterned media: viable route to 50Gb/in2 and up for magnetic recording," *IEEE Transactions on Magnetics*, vol. 33, no. 1, pt. 2, pp. 990-995, Jan. 1997.

[WOL63] J. K. Wolf, "Error-locating codes–a new concept in error control," *IEEE Transactions on Information Theory*, vol. 9, no. 2, pp. 113-117, Apr. 1963.

[WOL65] J. K. Wolf, "On codes derivable from the tensor product of check matrices," *IEEE Transactions on Information Theory*, vol. 11, no. 2, pp. 281-284, Apr. 1965.

[WWZK84] J. K. Wolf, A. D. Wyner, J. Ziv, and J. Kőrner, "Coding for a write-once memory," *AT&T Bell Labs, Technical Journal,* vol. 63, no. 6, 1984, pp. 1089-1112.

[WOL06] J. K. Wolf, "An introduction to tensor product codes and applications to digital storage systems," *IEEE Information Theory Workshop*, Punta del Este, Uruguay, Oct. 2006.

[WWKM09] R. Wood, M. Williams, A. Kavcic, and J. Miles, "The feasibility of magnetic recording at 10 terabits per square inch on conventional media," *IEEE Transactions on Magnetics*, vol. 45, no. 2, pp. 917-923, 2009.

[WU10] Y. Wu, "Low complexity codes for writing write-once memory twice," *IEEE International Symposium on Information Theory*, Austin, TX, Jun. 2010.

[YSVW11] E. Yaakobi, P.H. Siegel, A. Vardy, and J.K. Wolf, "On codes that correct asymmetric errors with graded magnitude distribution," *IEEE International Symposium on Information Theory*, St. Petersburg, Russia, Aug. 2011.

[YGSSW12] E. Yaakobi, L. Grupp, P.H. Siegel, S. Swanson, and J.K. Wolf, "Characterization and error-correcting codes for TLC Flash memories," *IEEE International*

Conference on Computers, Networks and Communications, Maui, HI, Jan.-Feb. 2012.

[YKSVW12] E. Yaakobi, S. Kayser, P. H. Siegel, A. Vardy, and J. K. Wolf, "Codes for write-once memories," *IEEE Transactions on Information Theory*, vol. 58, no. 9, Sep. 2012, pp. 5985-5999.

[YSVW12] E. Yaakobi, P. Siegel, A. Vardy, and J. Wolf, "Multiple error-correcting wom-codes," *IEEE Transactions on Information Theory*, vol. 58, no. 4, pp. 2220-2230, 2012.

[YS12] E. Yaakobi, A. Shpilka, "High sum-rate three-write and non-binary WOM codes," submitted to *IEEE Transactions on Information Theory*, 2012.

[ZJB11] H. Zhou, A. Jiang, and J. Bruck, "Nonuniform codes for correcting asymmetric errors," *IEEE International Symposium on Information Theory*, St. Petersburg, Russia, Aug. 2011.

[ZJB12] H. Zhou, A. Jiang, and J. Bruck, "Systematic error-correction codes for rank modulation," *IEEE International Symposium on Information Theory*, Cambridge, MA, Jul. 2012.