Spring 4-25-2014

Combinatorial and Algebraic Coding Techniques for Flash Memory Storage

Kathryn A. Haymaker
University of Nebraska-Lincoln, khaymake@gmail.com

COMBINATORIAL AND ALGEBRAIC CODING TECHNIQUES FOR FLASH MEMORY STORAGE

by

Kathryn Haymaker

A DISSERTATION

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Doctor of Philosophy

Major: Mathematics

Under the Supervision of Professor Christine A. Kelley

Lincoln, Nebraska

May, 2014

COMBINATORIAL AND ALGEBRAIC CODING TECHNIQUES FOR FLASH MEMORY STORAGE

Kathryn Haymaker, Ph. D.

University of Nebraska, 2014

Adviser: Christine A. Kelley

Error-correcting codes are used to achieve reliable and efficient transmission when storing or sending information across a noisy channel. This thesis investigates a mathematical approach to coding techniques for storage devices such as flash memory storage, although many of the resulting codes and coding schemes can be applied in other contexts. The main contributions of this work include the design of efficient codes and decoding algorithms using discrete structures such as graphs and finite geometries, and developing a variety of strategies for adapting codes to a multi-level setting.

Information storage devices are prone to errors over time, and the frequency of such errors increases as the storage medium degrades. Flash memory storage technology has become ubiquitous in devices that require high-density storage. In this work we discuss two methods of coding that can be used to address the eventual degradation of the memory.

The first method is rewriting codes, a generalization of codes for write-once memory (WOM), which can be used to prolong the lifetime of the memory. We present constructions of binary and ternary rewriting codes using the structure of finite Euclidean geometries. We also develop strategies for reusing binary WOM codes on multi-level cells, and we prove results on the performance of these strategies.

The second method to address errors in memory storage is to use error-correcting

codes. We present an LDPC code implementation method that is inspired by bit-error patterns in flash memory. Using this and the binary image mapping for non-binary codes, we design structured nonbinary LDPC codes for storage. We obtain performance results by analyzing the probability of decoding error and by using the graph-based structure of the codes.

DEDICATION

This dissertation is lovingly dedicated to my parents

Thomas and Ann Haymaker,

and in fond remembrance of

Dr. Jenna Higgins.

ACKNOWLEDGMENTS

This dissertation would not have been possible without the work of Professor Christine Kelley, who has generously provided me with guidance and support since my first week at UNL. I was fortunate to have such a dedicated and insightful mentor. Working with Professor Kelley helped me grow mathematically and professionally, and I will be forever grateful for the many opportunities that she gave me. I would also like to thank the members of my doctoral committee: Professor Judy Walker and Professor Jamie Radcliffe, for providing me with valuable feedback at many points during my graduate career, and Professor Myra Cohen and Professor David Pitts, for showing consistent interest in my work.

My family has given me immense support. This dissertation is dedicated to my parents, because their selfless love is beyond words. They would go to the end of the world for the sake of family, but instead they ended up repeatedly driving to Nebraska (and beyond). Thank you to my siblings Kelly and Joey, for always challenging me to improve by being better than me at many things. I am also grateful to my grandparents for providing me with a strong example of honest hard work, which has been particularly inspiring these past few years!

I would like to acknowledge my officemates at UNL: Abby, Kaelly, and Molly, for being excellent friends and making our first year cozy in our three-person office, and Simone for many thoughtful conversations during my last year. Molly has also been a wonderful roommate, co-teacher, and work partner. Thank you to Ashley and Melanie, for work time and crossword time, and everything in between. Your friendship means so much to me. The UNL Department of Mathematics provides a wonderful environment for graduate students, and I have enjoyed learning and teaching alongside this group of talented individuals. I owe particular thanks to

Amanda, Anisah, Ben, Courtney, James, Lauren, and Sara for being amazing, each in amazingly different ways. I would also like to acknowledge the other members of RAD for making our "recreational research" so much fun, even in the midst of realizing we had been scooped. Thank you to Bo Zhang and Angela Bliss for providing helpful comments throughout the past two years of the dissertation writing process.

Professor Rhonda Hughes of Bryn Mawr College first inspired me to pursue mathematics and continues to provide inspiration. I am also grateful to Professor Paul Melvin and Professor Leslie Cheng for their encouragement and support. Thank you to my dear friends from Bryn Mawr—particularly Alice, Talia, Nicole, and Jenna.

Finally, I would like to thank Nathan Corwin for infusing my life with an abundance of humor and love.

GRANT INFORMATION

This work was supported in part by the National Security Agency under Grant Number H98230-11-1-0156. The United States Government is authorized to reproduce and distribute reprints not-withstanding any copyright notation herein. This work was also supported in part by a University of Nebraska Presidential Fellowship, the United States Department of Education GAANN grant number P200A090002, and an NSF EPSCoR First Award.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Coding theory has a rich history of drawing motivation from questions that arise in applications. This has traditionally occurred in the context of sending information across a communication channel, a system where the output depends probabilistically on the input. The channel either delivers the information to another point in space (transmission) or another point in time (storage). The *Shannon capacity* of a channel is the maximum transmission rate that a code family can achieve while a fixed decoder returns an error with probability approaching zero. This is the theoretical 'best' that a code can do on a given communication channel.

This thesis explores coding techniques motivated by storage applications, focusing particularly on motivation from the structure of flash memory. The encoding schemes and analysis that we present can be applied in a variety of settings, and we also address classical questions in the context of modern code constructions.

Write-once memory (WOM) codes were developed in the 1980s as a method of reusing write-once memory media, such as punch cards. Both the WOM model and the flash memory model share an asymmetric write/erase property that allows for rewriting with certain restrictions. In a WOM, information is stored in the form of

a binary vector. The current state of the memory can be changed, or rewritten, to represent a new message at a later time, with the restriction that a 'one' that has been written in the memory cannot be changed back to a 'zero'. Flash memory has a similar constraint. The information is encoded and stored in the memory in blocks of cells, where each cell can be charged up to one of q levels. The process for increasing charge allows a single cell to be increased at a time, but to decrease the charge of a cell, the entire block containing that cell must be erased and reprogrammed. After many erasures, the quality of the memory begins to degrade. Thus, it is advantageous to rewrite on the same space in the memory as many times as possible, always increasing the cell levels, before requiring an erasure. This can be viewed as a mapping from a given sequence of information vectors at time $t = 1, 2, \ldots$ to a sequence of distinct component-wise monotonically increasing memory states, where the decoding is given by the inverse map. Contrary to WOM, flash memory allows for decreases in cell charge, but at a long-term cost in reliability. We refer to WOM codes and q-ary generalizations collectively as *rewriting codes*. An important feature of this asymmetric write/erase model is that the information stored at a given time t need not be retained at subsequent time-steps.

A notable difference between error-correcting codes and rewriting codes is that rewriting codes are generally not linear codes, and they often rely on ad hoc mapping schemes that are specific to the parameters of the construction. In this thesis we take a mathematical approach to creating families of codes with explicit mapping schemes using discrete structures. First, we introduce infinite WOM code families based on finite geometries. Moreover, we present new multi-level coding schemes for rewriting codes that have flexible component codes to fulfill a variety of parameters. These approaches include concatenation involving error correction and rewriting, and generalizations of the coset encoding and position modulation schemes for write-once

memories. The coset encoding scheme relies on knowledge about the covering radius of an error-correcting code, and to that end we also find bounds on the covering radius of codes based on the incidence structure of finite geometries.

Since all memory devices are susceptible to errors, it is important to consider how error-correcting codes should be implemented in the memory. Families of long blocklength low-density parity-check (LDPC) codes used with iterative decoding algorithms are excellent candidates for error-correction in storage. We use a common representation of these codes—a sparse bipartite (Tanner) graph. We perform an iterative analysis based on the probability of decoding failure, using the degree distribution of the underlying graph. These general ideas are applied to LDPC codes in MLC (four-level) flash memory. In this thesis we give a description of optimal code implementation that uses the check node degrees, and in particular we show that the standard scheme of bit-interleaved coded modulation results in the worst case implementation of an LDPC code in MLC flash memory. We also demonstrate how to choose the check node connections to the types of variable nodes so that the performance is optimized.

This thesis is focused on three general areas of study—codes using the structure of finite geometries, multi-level coding schemes using component codes, and the implementation of graph-based codes in flash memory. Our new construction of a family of WOM codes that utilizes the incidence structure of finite geometries benefits from simple encoding and decoding descriptions based on the incidence of the geometry, and from blocklengths that are powers of two. We also derive bounds on the covering radius of LDPC codes constructed from finite geometries. We then present several methods of combining rewriting codes to create a diverse and applicable collection of coding techniques for multi-level flash memory. Finally, we present a framework for studying the application of graph-based error-correcting codes to a storage set-

ting in which different bit-error probabilities can be identified in the memory. By synthesizing these areas of study and using ideas from applied discrete mathematics, we provide a novel approach to two of the major directions in coding for storage: rewriting codes, and the application of error-correcting codes.

The thesis is organized as follows. Chapter 2 introduces the necessary background in coding theory and the flash memory application. In Chapter 3 we use the incidence structure of finite geometries to create encoding and decoding maps for WOM codes and ternary flash codes, and we provide proofs of the parameters of the resulting codes. Chapter 4 presents methods of designing flash codes, including the methods of generalized position modulation, concatenation involving both error-correction and increased rewriting, and a generalization of the binary coset encoding scheme to multi-level cells. We provide parameters for the resulting codes and give examples of the large variety of code families that result from these construction methods. Chapter 5 explores the application of binary LDPC codes to flash memory with $q = 4$ levels, where the memory contains two distinct channel bit-error probabilities. We analyze the probability of decoding error for the Gallager A and B decoding algorithms, and we determine the optimum configuration of coded bits to positions in the memory. In Chapter 6 we use the binary image of a code over $\mathbb{F}_4$, along with insights from Chapter 5 to determine nonbinary edge labels for $(3, 6)$-regular LDPC codes. We analyze these configurations using binary decoding on the binary expanded graph, and we also use nonbinary Gallager-type hard-decision decoding to assess the performance of the edge label sets. Chapter 7 contains bounds on the covering radius of finite geometry LDPC codes, which show that in general the covering radius of these codes grows with the field size of the underlying geometry. Chapter 8 concludes the thesis.

Chapter 2

Preliminaries

Since the time of telegraphs in the 19th century, people have attempted to create reliable ways of sending messages across a noisy channel [4]. However, Claude Shannon was the first person to formalize this study and place it on firm mathematical footing [67]. A *communication channel* is a collection of triples: an input, an output, and a transition probability. The larger context of a digital communication system is shown in Figure 2.1.

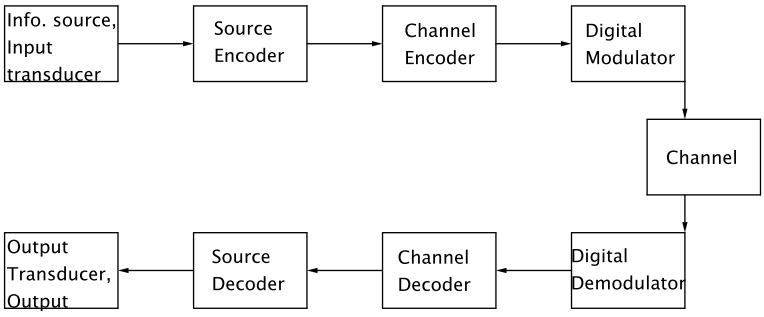

Figure 2.1: A model of a digital communication system [58].

Channel coding is concerned with adding redundancy to information in a structured way so that after modulation, channel transmission, and demodulation, the

original message can be recovered. Structure is needed to ensure that the encoding and decoding processes can be accomplished in a practical and efficient way.

Figure 2.2 shows a simple box diagram of the process of information encoding and decoding. The main channel that we are concerned with in this thesis is the flash memory storage channel. Some common channel models that we will use to model the physical system include the binary symmetric channel (BSC) and the Additive White Gaussian Noise (AWGN) channel.

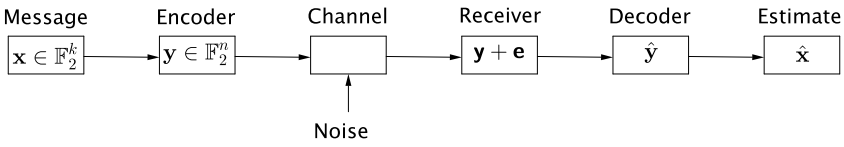

Figure 2.2: A model of channel coding for transmission over a binary memoryless channel.

The BSC has input and output alphabet $\{0, 1\}$, and has a crossover probability p. Figure 2.3 shows this channel.

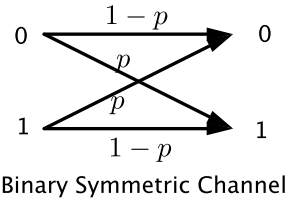

Figure 2.3: The binary symmetric channel with crossover probability p.

Every communication channel has an associated parameter called the *channel capacity*, which captures the maximum rate at which information can be sent reliably

across the channel[1]. The BSC has capacity

$$\mathcal{C} = 1 + p\log(p) + (1-p)\log(1-p).$$

The AWGN channel is another common channel model. Rather than flipping bits, the noise in this case is an additive model, where if $\mathbf{x}$ is sent, then $\mathbf{y} = \mathbf{x} + \mathbf{n}$ is received, where $\mathbf{n}$ is a vector capturing the noise in the channel. The model reflects natural occurrences of noise that can perturb the transmitted symbol by a continuous rather than a discrete amount. Each entry of the noise vector $\mathbf{n}$ is independent and identically distributed, with a normal distribution with zero mean and variance σ^2. The continuous values are then mapped to discrete symbols at the receiver.

Shannon showed that the essential limit on communication comes in the form of time rather than reliability. Shannon's *Noisy Channel Coding Theorem* is as follows:

Theorem 2.0.1 (Shannon, 1948)**.** *Given a channel with capacity $\mathcal{C}$, for any $\epsilon > 0$ and $R < C$, for large enough N, there exists a code of length N and rate at least R and a decoding algorithm, such that the maximal probability of block error is smaller than ϵ.*

Shannon's Noisy Channel Coding theorem shows that coding can be used to transmit information over a noisy channel at any rate below the channel capacity within a desired probability of decoding error.

2.1 Error-correcting codes

A *linear error-correcting code* $\mathcal{C}$ is a subspace of a finite-dimensional vector space over a finite field $\mathbb{F}_q$. The dimension n of the vector space is the *blocklength* of the

[1]While such a parameter always exists, the exact value may not be known.

code. The dimension k of the subspace is the number of information symbols. The *Hamming distance* between two vectors is the number of positions in which they differ. The minimum Hamming distance between any two distinct codewords in $\mathcal{C}$ is denoted by d. To correct t Hamming errors and decode to the nearest codeword, it is necessary to have $d \geq 2t + 1$. Thus, codes with large minimum distance perform well under "nearest neighbor" decoding algorithms. The *relative minimum distance* of a code is $\frac{d}{n}$. For a code $\mathcal{C}$ with blocklength n, dimension k, and minimum distance d, we say that $\mathcal{C}$ is an $[n, k, d]$ code.

A code family $\{\mathcal{C}_i\}$ is called *asymptotically good* if both the rate and the relative minimum distance are bounded away from zero in the limit as $i \to \infty$.

Since $\mathcal{C}$ is a subspace, we can use a matrix to define the code. A *generator matrix* is a matrix G whose image is the code. A *parity check matrix* is a matrix H whose *kernel* is $\mathcal{C}$, i.e., $\mathbf{v} \in \mathcal{C}$ if and only if $H\mathbf{v}^T = \mathbf{0}$. For $\mathbf{v} \in \mathbb{F}_q^n$, the vector $\mathbf{u} = H\mathbf{v}^T$ is the *syndrome* of $\mathbf{v}$. If a transmitted codeword results in a nonzero syndrome at the receiver, then at least one error has occurred (a zero syndrome does not guarantee perfect transmission, however).

The *covering radius* of an error-correcting code $\mathcal{C} \subseteq \mathbb{F}_q^n$ is the smallest integer $\mathcal{R}$ such that Hamming spheres of radius $\mathcal{R}$ centered at codewords cover the space $\mathbb{F}_q^n$. The covering radius is a parameter that is difficult to determine in general, but it is feasible to obtain bounds on the covering radius of particular families of codes. Equivalently,

$$\mathcal{R}(\mathcal{C}) = \max_{x \in \mathbb{F}_2^n} \min_{c \in \mathcal{C}} d(x, c).$$

The covering radius of classical binary linear codes has been studied extensively, and many of the known results are contained in the reference [9].

An $[n, k, d]$ code is called *perfect* if $\lfloor \frac{d-1}{2} \rfloor = \mathcal{R}(\mathcal{C})$. In this case, spheres of radius

$\mathcal{R}(\mathcal{C})$ cover the space with no overlap; that is, every vector in the the space is contained in exactly one sphere around a codeword.

2.1.1 Hamming codes

In 1948, Richard Hamming introduced the first explicit construction of a code family, now called *Hamming codes* [22]. They are perfect codes with minimum distance three and parameters $[2^m - 1, 2^m - m - 1, 3]$, for $m \geq 2$. A binary Hamming code of length $2^m - 1$ is determined by a parity-check matrix where the columns are precisely all nonzero binary vectors of length m.

The following parity-check matrix defines the $[7, 4, 3]$ Hamming code.

$$H = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

The three rows of H determine the following parity check equations. Since the code is the nullspace of the parity-check matrix H, codewords are precisely the $(x_1, \ldots, x_7) \in \mathbb{F}_2^7$ such that

$$x_4 + x_5 + x_6 + x_7 = 0 \,(\mathrm{mod}\, 2)$$
$$x_2 + x_3 + x_6 + x_7 = 0 \,(\mathrm{mod}\, 2)$$
$$x_1 + x_3 + x_5 + x_7 = 0 \,(\mathrm{mod}\, 2).$$

For q a power of a prime, a Hamming code over $\mathbb{F}_q$ is determined by a parity-check matrix with columns all nonzero vectors of length m over $\mathbb{F}_q$, such that the first entry is 1. For any q a power of a prime and $m > 1$, there is a Hamming code with parameters $[\frac{q^m-1}{q-1}, \frac{q^m-1}{q-1} - m, 3]$. Hamming codes can be easily decoded using

syndrome decoding.

2.1.2 Reed-Muller codes

The Reed-Muller family of codes is another early construction of error-correcting codes. There are various combinatorial descriptions of the codes [50], including methods involving the Kronecker product of matrices, vector concatenation, Boolean algebras, and binary exponentiation [65] (the method described below). The original code family was introduced by Muller in [53]; Reed [59] devised a decoding algorithm for the codes. The binary r^{th} order Reed-Muller code of length 2^m is denoted by $\mathcal{R}(r, m)$, where $m \in \mathbb{N}$ and $0 \leq r \leq m$.

Let $S(r, m) \subseteq \mathbb{F}_2^m$ be the set of binary vectors of length m with Hamming weight at most r (i.e., the sphere of radius r centered at $\mathbf{0}$).

$$|S(r, m)| = \sum_{j=0}^{r} \binom{m}{j}.$$

The code $\mathcal{R}(r, m)$ can be defined by a generator matrix $G_{RM}(r, m)$ that has rows indexed by elements of $S(r, m)$ and columns indexed by vectors in $\mathbb{F}_2^m$. The entry in the matrix indexed by the pair $(\mathbf{e}, \mathbf{a})$ is 0 if there exists at least one index $i \in \{0, \ldots, m-1\}$ such that $a_i = 0$ and $e_i = 1$. Otherwise the entry is 1. The resulting code has parameters $[2^m, |S(r, m)|, 2^{m-r}]$. For a fixed $m > 0$, the following inclusions hold:

$$\mathcal{R}(0, m) \subset \mathcal{R}(1, m) \subset \cdots \subset \mathcal{R}(m, m).$$

The code $\mathcal{R}(0, m)$ consists of two codewords: $\mathbf{0}, \mathbf{1} \in \mathbb{F}_2^{2^m}$ (it is the binary repetition code of length 2^m). $\mathcal{R}(m, m)$ consists of all even weight words in $\mathbb{F}_2^{2^m}$.

The standard decoding method for Reed-Muller codes is majority-logic decoding

[59], a process that decodes subsets of bits based on the majority value, then uses this to iteratively deduce the values of larger subsets of bits. Majority-logic decoding can be practically implemented in applications with circuits.

2.1.3 LDPC codes

Low-density parity-check (LDPC) codes were introduced by Robert Gallager in his 1963 thesis [20] and in [19]. The codes and iterative decoding methods that Gallager discussed were rediscovered several times over the years, notably by Tanner in 1981 [73], but it wasn't until the mid-1990s when the computational power for iterative decoding was available that the wider research community became fully aware of the potential for capacity-approaching families of LDPC codes with efficient iterative decoders[2].

An *ensemble of LDPC codes* over $\mathbb{F}_q$ is a family of linear codes with sparse parity-check matrices. In this case, *sparsity* means that as $mn_i \to \infty$ (where n_i represents the code lengths), there is a constant c such that there are fewer than $c\max\{m, n_i\}$ ones in the matrix [69]. Denote by $\mathcal{C}(H)$ a code determined by a parity-check matrix H.

Gallager introduced a family of binary LDPC codes, analyzed their distance properties, and presented an iterative decoding procedure for the codes [19]. In this subsection, we review Gallager's original construction to give an example of an LDPC code ensemble.

Gallager's construction consists of families of (j, k)-*regular low-density codes*, where each column of each parity check matrix has j ones, and each row has k ones. Let n denote the blocklength of a particular code, and let K denote the dimension. There

[2]Gallager's paper was cited about 80 times during the years 1962-1995. The total number of citations from 1962-2014 exceeds 8400.

are $m = \frac{nj}{k}$ rows in the parity check matrix H. Here, $\frac{n}{k}$ is an integer, and $\frac{n}{k} = \frac{m}{j}$.

The construction begins with H^*, an $(\frac{n}{k} \times n)$ matrix where each row consists of k ones, as follows:

$$H^* = \begin{bmatrix} 1 & \cdots & 1 & 0 & 0 & \cdots & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 1 & \cdots & 1 & 0 & \cdots & 0 \\ \vdots & & & \ddots & & & & & \vdots \\ 0 & \cdots & & & & 0 & 1 & \cdots & 1 \end{bmatrix}$$

A parity check matrix of a low-density code in the (j, k) family is formed by taking random permutations of the columns, denoted $\sigma_i(H^*)$, of this matrix and stacking them:

$$H = \begin{bmatrix} H^* \\ \hline \sigma_1(H^*) \\ \hline \vdots \\ \hline \sigma_{j-1}(H^*) \end{bmatrix}$$

Define (j, k)-*Gallager codes* to be the ensemble of codes obtained over all random permutation of the columns of H^* in the bottom $j - 1$ submatrices, where each permutation is assigned equal probability.

A *Tanner graph* for $\mathcal{C}(H)$ is a bipartite graph with vertices $U \cup V$ whose incidence matrix is H. The columns of H correspond to the vertex set U, known as *variable nodes*, and the rows of H correspond to the vertex set V, or *check nodes*. If the i, j^{th} entry in H is $\gamma \neq 0$, it results in an edge labeled γ in the Tanner graph[3]. If the entry is zero there is no edge in the Tanner graph. A vector $(v_0, \ldots, v_{n-1}) \in \mathbb{F}_q^n$ is in the code $\mathcal{C}(H)$ if and only if for every check node c the linear combination of neighbors of c with coefficients given by edge labels is zero in the field.

[3]If $\gamma = 1$ no edge label is used since 1 is implied.

Example 2.1.1. Take $n = 16$, $j = 3$, and $k = 4$. The following is a parity-check matrix for a $(3, 4)$-Gallager code of length 16, which determines a $[16, 6, 6]$ binary code. The rank of H is 10.

$$H = \left[\begin{array}{cccc|cccc|cccc|cccc} 1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1 \\ \hline 1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0 \\ 0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0 \\ 0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0 \\ 0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1 \\ \hline 1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1 \\ 0&1&0&0&0&0&1&0&0&0&0&1&1&0&0&0 \\ 0&0&1&0&0&0&0&1&1&0&0&0&0&1&0&0 \\ 0&0&0&1&1&0&0&0&0&1&0&0&0&0&1&0 \end{array}\right].$$

□

A Tanner graph constructed from H is given in Figure 2.4.

Fix a parity check matrix H belonging to the (j, k)-Gallager ensemble. First, note that the dimension K of the code C that is the nullspace of H satisfies the following bound:

$$K \geq n - m.$$

Recall that $m = \frac{nj}{k}$. Thus, we get

$$K \geq n - \frac{nj}{k} = n(1 - \frac{j}{k}).$$

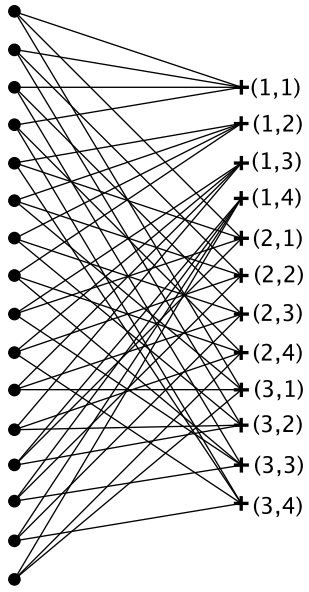

Figure 2.4: Tanner graph for $\mathcal{C}$.

Therefore the rate of the code satisfies:

$$\frac{K}{n} \geq 1 - \frac{j}{k}.$$

LDPC codes (along with message-passing decoding) have emerged as the first class of codes to approach capacity [62, 60]. This remarkable result was shown roughly 50 years after Shannon's work [67] and after many earlier sophisticated constructions of codes. The simple definition of these codes allows for the construction of random ensembles of codes, and in fact random families of (j, k)-regular LDPC codes are provably asymptotically good for $j \geq 3$ [19, 10]. An explicit family of asymptotically good LDPC codes based on expander graphs were presented in [71]. Some important features of code design are the degree distribution on the Tanner graph, which has an impact on the decoding threshold [48], and the minimum distance of the code. The typical performance simulation of a code plots SNR (signal-to-noise ratio) versus

the frame error rate. In this setting the error floor is the region where the "waterfall curve" begins to flatten out as the SNR increases (i.e., the decoding performance does not improve with better channel quality). Richardson [63] observed that error floors of LDPC codes are often the result of near-codewords, and Kelley and Sridhara later gave a characterization of such occurrences in terms of the Tanner graph [39]. Study of the error floor effect and the design of structured LDPC codes have become two important topics in modern coding theory. Moreover, LDPC codes are currently used in many applications requiring reliable codes with good rate and efficient decoding.

2.2 Coding for flash memories

As data creation and usage proliferates, digital storage media is becoming increasingly important. Storage technology must be fast, reliable, and have high storage capacity. Flash memory is a type of non-volatile memory device, meaning that the information is retained even when the power source is removed. The codes and techniques in this thesis were inspired by the structure of flash memory, but the ideas have broad applications in storage technologies. Examples of other types of storage media include magnetic recording, phase-change memory, and millipede memory, a nanotechnology version of a punch card.

2.2.1 Flash memory structure

Flash memories are useful due to their potential for high storage capacity and low power consumption. Flash memory storage is a technology that is based on organizing the memory into blocks of cells in which each cell can be charged up to one of q levels. While increasing the charge of a cell is easy, decreasing the charge is costly since the entire block containing the cell must be erased and rewritten. Such an operation

involves reprogramming roughly 10^5 cells. Moreover, frequent block erasures also reduce the lifetime of the flash device. It is therefore desirable to be able to write as many times as possible before having to erase a block [64, 14, 38, 5]. Like any storage device, the flash cells are also prone to errors due to charge leakage or the writing process. Thus, the coding design goals for flash memories include maximizing the number of writes between block erasures, correcting cell charge leakage errors, and correcting errors that occur during the writing process.

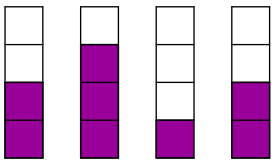

Figure 2.5: Model of flash memory cells holding charges $(2, 3, 1, 2)$.

An information theoretic approach to writing on memories with defects was first considered by Kuznetsov and Tsybakov [45], and later surveyed in [46]. These binary defects are commonly in the form of a "stuck-at bit", meaning that a bit in the memory is either stuck at the value zero or one. The write-once memory (WOM) model, introduced by Rivest and Shamir [64], and other constrained memory models (WUM, WIM, WEM[4]) can be considered as particular cases of the general defective channel [46, 1, 7], where the positions with ones are regarded as 'defects' for the second write, since the ones cannot be changed back to zeros. Although WOM codes—first motivated by punch cards—were studied extensively in the 1980s [64, 83], interest in these rewriting schemes continued through the 1990s [16], and was renewed in 2007 due to the notable link to flash memory applications observed by Jiang [29]. Due to

[4]WUM, WIM, and WEM stand for write-unidirectional, write-isolated, and write-efficient memory, respectively.

the asymmetric costs associated with increasing and decreasing cell levels, the flash memory model can be viewed as a generalization of the WOM model.

As a result, new constructions of binary WOM codes have been proposed for flash cells having two levels (i.e., capable of storing one bit of information per cell) [29, 32, 82], including some capacity-achieving schemes [70]. Error-correcting codes for the general defective channel and for WOM have also been considered, although addressing errors while incorporating rewriting capabilities is difficult, and many codes in the literature are optimized primarily for one of these goals [5, 35, 33, 32, 84, 29]. Next we present the terminology and notation for WOM and flash codes that will be used in this thesis.

2.2.2 WOM codes

A write-once memory (WOM) is a storage device over a binary alphabet where a zero can be increased to a one, but a one cannot be changed back to a zero. An information message is encoded and stored in a string of cells in the memory, referred to as a *cell state vector*[5]. The cells in the cell state vector form the symbols of the codeword and can be updated, or *rewritten*, to yield a new cell state vector representing a different message.

A write-once memory code is composed of a set V of information words and a set S of cell state vectors with $S \subseteq \mathbb{F}_2^n$, corresponding to the codewords of the WOM code. Many different cell state vectors can represent the same information message. In addition, the WOM code is equipped with an encoding and decoding function. The encoding function takes as inputs both the current state of the memory and the new information message to be stored. Specifically, it maps the current cell state vector

[5]This terminology was introduced in [29] in reference to the structure of flash memory, but it is convenient to use in the WOM case as well.

to an updated cell state vector that represents the new information message and is component-wise greater than or equal to the previous state. The decoding function maps the resulting cell state vector to the updated information message. Only the most recently written message is retained. The amount of information messages that can be encoded at each time step need not be the same, however, as the following notation conveys.

Definition 2.2.1. Let $\langle v_1, \ldots, v_t \rangle / n$ denote a t-write *variable-rate WOM code* on n cells, where v_i is the number of messages that can be represented on the i^{th} write. In the fixed information case, i.e., when $v_1 = \cdots = v_t$, a *t-write WOM code* will be denoted by $\langle v \rangle^t / n$, and be called a *fixed-rate WOM code*.

The *sum-rate* (or simply *rate*) of a WOM code is

$$R = \frac{\log_2(v_1 \cdots v_t)}{n}.$$

The next example, from [64], is the canonical example of a WOM code.

Example 2.2.2. The Rivest and Shamir WOM code is shown in Table 2.2.2 [64]. It maps two information bits to three coded bits and is capable of tolerating two writes. It has rate $\frac{\log_2(16)}{3} = \frac{4}{3}$. Any of the four messages may be written at either write. The table is interpreted as follows: on the first write, the encoding function takes the current all-zero state and the new information message $\mathbf{v}$ and maps it to the representation of $\mathbf{v}$ in the 'first write' column. On the second write, the encoding function takes the current cell state and the new information message $\mathbf{v}'$ and outputs the cell state vector opposite $\mathbf{v}'$ in the 'second write' column. For example, the message sequence $01 \rightarrow 11$ would be recorded as $100 \rightarrow 110$. If the new information message is the same as the information represented by the current cell state vector, the

Information	1^{st} write	2^{nd} write
00	000	111
01	100	011
10	010	101
11	001	110

Table 2.1: $\langle 4 \rangle^2/3$ WOM-code by Rivest and Shamir.

memory remains unchanged. Decoding is as follows: the cell state vector (a_1, a_2, a_3) can be decoded as $((a_2 + a_3) \pmod 2 \, , (a_1 + a_3) \pmod 2)$.

□

2.2.3 Flash Codes

When $q = 2$, the flash cell is called a single level cell (SLC) since the cell can only represent one nonzero value, and a multi-level cell when $q > 2$. Since flash memory applications often have $q = 4$, we will use multi-level cell (MLC) to mean specifically $q = 4$ in Chapters 5 and 6. An SLC can store one bit of information per cell whereas an MLC with $q = 4$ can store two bits of information per cell. Fiat and Shamir considered a generalized version of a WOM, in which the storage cells have more than two states with transitions given by a directed acyclic graph [14]. The idea of extending to multi-level cells was further explored by Jiang in [29], in which he considered generalizing error-correcting WOM codes. Techniques for rewriting codes on q-ary cells include floating codes, which were introduced by Jiang, Bohossian, and Bruck [30], and more generally, trajectory codes, which are described in [34]. Although these are similar objects (i.e., mapping schemes for rewriting), we will use the term *flash codes*, introduced in [82], to refer to a rewriting code on multi-level cells.

Definition 2.2.3. When $q > 2$, $\langle v \rangle_q^t / n$ will denote a t-write *flash code* for use on n cells having q levels, where v messages can be represented at each write. The capacity of a flash memory is the maximum amount of information that can be stored per cell with q levels for t writes.

Fu and Han Vinck [16] proved the following theorem on the theoretical limit on the rate of a flash code.

Theorem 2.2.4 (Fu, Han Vinck, 1999)**.** *The maximum total number of information bits that can be stored per q-ary cell over t writes is*

$$\log_2(1 + (q-1)t).$$

This gives that the best rate possible for a binary WOM code with two writes is $\log_2(3)$. The Rivest-Shamir code in Example 2.2.2 is approximately 0.252 from the best possible rate.

Chapter 3

Write-once memory codes from finite geometries[1]

In this chapter, we review an early construction of WOM codes from finite projective geometries and we present new constructions of both binary and ternary WOM codes from finite Euclidean geometries. These constructions have simple encoding and decoding maps, and they yield a wide variety of blocklengths for codes that can be used in multi-level flash coding schemes, to be discussed in Chapter 4.

3.1 Finite geometries

Finite geometries are incidence structures consisting of a set of points and subsets of points that define incidence relations. Here we present relevant definitions and examples of finite Euclidean and finite projective geometries. Further details are available in [2, 50].

[1]Material in this chapter has appeared in [25], *Designs, Codes and Cryptography* (Section 3.3), and in [24], the *Proceedings of the Asilomar Conference on Signals, Systems, and Computing* (Section 3.4).

Definition 3.1.1. The *m-dimensional Euclidean geometry over* $\mathbb{F}_2$, denoted by $EG(m, 2)$, is an incidence structure with 2^m points and $2^{(m-1)}(2^m - 1)$ lines. The set of points in $EG(m, 2)$ may be regarded as all m-tuples over $\mathbb{F}_2$, and each pair of points defines a unique line.

For $m > 0$ and p a prime, $EG(m, p^s)$ is the *m-dimensional Euclidean geometry over* $\mathbb{F}_{p^s}$. Points are in correspondence with m-tuples over $\mathbb{F}_{p^s}$. The vector space structure of the m-tuples over $\mathbb{F}_{p^s}$ can be used to define the incidence structure of the geometry. A *μ-flat* is a μ-dimensional subspace of the vector space or a coset of such a subspace. For example, 1-flats are lines, 2-flats planes, and $(m-1)$-flats are called *hyperplanes*.

Similarly, $PG(m, p^s)$ is the *m-dimensional projective geometry over* $\mathbb{F}_{p^s}$. Points in the geometry are in correspondence with one-dimensional subspaces of the vector space of $(m+1)$-tuples over $\mathbb{F}_{p^s}$.

Since the constructions in the following sections deal with $q = 2$, we give a more specific description of the geometries $EG(m, 2)$ and $PG(m, 2)$.

Let X be the set of points in $EG(m, 2)$. A μ-flat in $EG(m, 2)$ passing through a point a_0 consists of points of the form $a_0 + \beta_1 a_1 + \cdots + \beta_\mu a_\mu$, where $a_0, \ldots, a_\mu \in X$ are linearly independent and $\beta_1, \ldots, \beta_\mu \in \mathbb{F}_2$.

The number of μ-flats in $EG(m, 2)$ is

$$2^{(m-\mu)} \prod_{i=1}^{\mu} \frac{2^{(m-i+1)} - 1}{2^{(\mu-i+1)} - 1}.$$

Moreover, each μ-flat in $EG(m, 2)$ is a coset of a $EG(\mu, 2)$, and thus contains 2^μ points.

Definition 3.1.2. The *finite projective geometry of dimension m over* $\mathbb{F}_2$, denoted

$PG(m,2)$, is an incidence structure with $2^{m+1}-1$ points and $\frac{(2^{m+1}-1)(2^m-1)}{3}$ lines. The points are the nonzero $(m+1)$-tuples $(a_0, a_1, \ldots, a_m) \in \mathbb{F}_2^{m+1}$, and a line through two distinct points a_0 and a_1 contains exactly the set of points $\{a_0, a_1, a_0 + a_1\}$.

Example 3.1.3. $PG(2,2)$ is the 2-dimensional finite projective geometry over $\mathbb{F}_2$, known as the *Fano plane*. It has seven points, labeled 1-7, and seven lines, as shown in Figure 3.1. Each line contains three points, and each point lies on exactly three lines. □

3.2 The Merkx construction

In 1984, Merkx constructed a family of WOM codes based on the m-dimensional finite projective geometries over $\mathbb{F}_2$ [52]. The construction exploits a connection between the binary Hamming codes and $PG(m,2)$ that allows the WOM codes to be decoded easily via syndrome decoding. Specifically, Merkx uses the fact that the minimum weight codewords of the $[2^{m+1}-1, 2^{m+1}-m, 3]$ Hamming code $\mathcal{C}$ generate $\mathcal{C}$ and correspond to the incidence vectors of lines in $PG(m,2)$ (see [50], e.g.).

For example, in Figure 3.1 incidence vectors of lines in the Fano plane correspond to the minimum weight nonzero codewords of the $[7,4,3]$ Hamming code presented in Section 2.2.1.

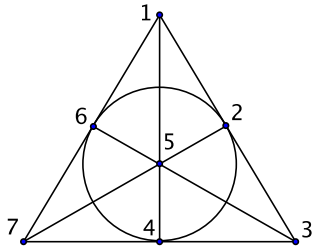

Figure 3.1: $PG(2,2)$ with labels that correspond to the $[7,4,3]$ Hamming code.

The nonzero minimum weight words in the $[7, 4, 3]$ Hamming code corresponding to the incidence vectors of lines are given in the following array.

P_1	P_2	P_3	P_4	P_5	P_6	P_7
0	1	0	1	0	1	0
1	0	0	0	0	1	1
1	1	1	0	0	0	0
1	0	0	1	1	0	0
0	0	1	1	0	0	1
0	0	1	0	1	1	0
0	1	0	0	1	0	1

In Merkx's construction, the messages correspond to points in the geometry. The WOM codewords, i.e. the cell state vectors, are a subset of $\mathbb{F}_2^{m+1} \setminus \mathcal{C}$, and thus, since the Hamming code is perfect, these codewords are always one error away from a binary Hamming codeword. The location of the error indicates the point in the geometry that corresponds to the information message.

Example 3.2.1. The $PG(2, 2)$ WOM code of [52] is a $\langle 7 \rangle^4/7$ code. Each position of a codeword corresponds to a point of the Fano Plane, and each codeword is the incidence vector of a substructure of the geometry that highlights a particular point being represented. WOM codewords are incidences of the following: on the first write, a point on the Fano Plane; on the second write, a line missing a point; on the third write, the union of a line with an additional point; on the final write, either the union of two lines or the plane missing a point. To decode the WOM code, Merkx observed that syndrome decoding identifies the information message. Figure 3.2.1 shows the write sequence $3 \to 5 \to 7 \to 3$ using the $\langle 7 \rangle^4/7$ code from the Fano Plane.

The arrow indicates the information point and the corresponding cell state vector representing that information is listed below each write. Note that the sequence of cell state vectors is monotonically increasing in each component as the write iteration increases.

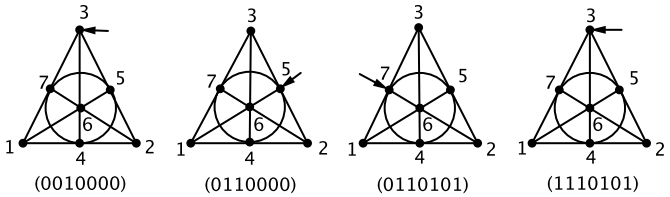

Figure 3.2: Four writes using the Merkx $PG(2,2)$ WOM code.

□

The following proposition by Cohen, Godlewski, and Merkx in [8] formulates more precisely the parameters of the WOM codes that result from this construction method.

Proposition 3.2.2 (Cohen, Godlewski, Merkx, 1986)**.** *For $m \geq 4$, the $[2^m - 1, 2^m - 1 - m]$ Hamming code yields a length $(2^m - 1)$ WOM code that can store m bits over $2^{m-2} + 2$ writes.*

3.3 WOM codes from $EG(m, 2)$

Since Hamming codes are punctured Reed-Muller codes, and are given by geometric designs over the binary field, we apply a similar construction strategy for designing WOM codes using $EG(m, 2)$. Minimum weight codewords of the r^{th} order Reed-Muller code of length 2^m, $\mathcal{R}(r, m)$, generate the code, and correspond to $(m-r)$-flats in the Euclidean geometry $EG(m, m-r)$. Analogous to the Merkx construction, we will use the connection between minimum weight words in $\mathcal{R}(m-2, m)$ and the planes

in $EG(m, 2)$ to construct our WOM code so that it inherits the easy decoding of the corresponding Reed-Muller code. We design the WOM codewords to be Hamming distance one away from a codeword of $\mathcal{R}(m-2, m)$. The WOM codewords are incidence vectors of configurations of points in the Euclidean geometry $EG(m, 2)$, including a point, a plane with a point missing, and the union of a plane with an additional point. These WOM codes may be decoded using any Reed-Muller decoding technique.

The next two examples illustrate this construction for $m = 3$ and $m = 4$.

Example 3.3.1. Using $EG(3, 2)$, the resulting code is an $\langle 8, 8, 8, 4\rangle/8$ WOM code. The code attains four writes on eight cells, where eight possible messages can be stored in the first three writes, and four messages can be stored in the fourth write. Recall that $EG(3, 2)$ has eight points, 28 lines, and 56 planes. Each message corresponds to one of the points in the geometry. On the first write, a message $i \in \{1, \ldots, 8\}$ is represented by a weight one cell state vector, where the one is in the i^{th} coordinate.

On the second write, the WOM codeword is a weight three cell state vector indicating a plane with a point missing, where the missing point is the information message. Since there are a several choices of planes containing the points i from the first write and the new message point j, the choice of plane can be made by putting an ordering on the points in the geometry and choosing the plane P containing both i and j which has a third point k that is smallest according to the ordering. Without this stipulation, the encoding process during the second write is not unique. Say that $P = \{i, j, k, k'\}$. After the second write, the cell state vector has weight three, with ones in positions i, k, k'.

On the third write, the ones in the cell state vector correspond to a plane union a point, where the additional point is the message[2] l. If l is not contained in the plane

[2]If $l = j$, then leave the contents of the cell state vector from the second write unchanged.

P from write two, then the cell state vector has ones in the positions corresponding to the four points in P and the position corresponding to l. If $l \neq j$ is contained in the plane P, then $l \in \{i, k, k'\}$, and there is a plane containing the *other* two points which does not contain l. Again, choose the plane that satisfied this requirement, and use the ordering on the points as indicated above. Observe that on each of the first three writes, it is possible to represent any of the eight messages.

Finally, on the fourth write, only messages corresponding to positions of the cell state vector with entry zero can be represented (except for the message represented in the third write, which can always remain on the fourth write, if needed). If i' is one of these messages, then to represent i' on the fourth write, the cell state vector will have a one in every coordinate except position i'.

As an example, the message sequence $1 \to 3 \to 2 \to 7$ is demonstrated in Figure 3.3.

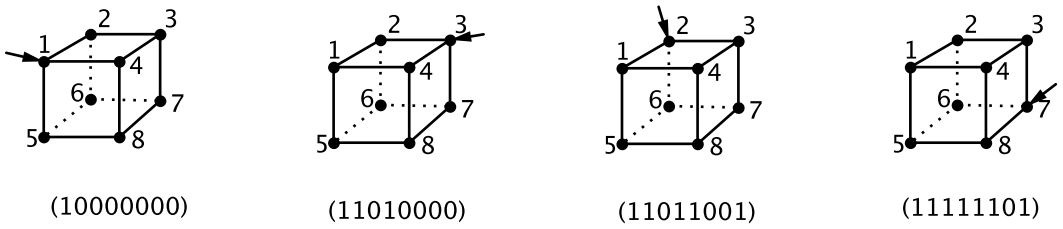

Figure 3.3: The message sequence $1 \to 3 \to 2 \to 7$ in the $EG(3, 2)$ WOM code.

□

In constructing the WOM code from $EG(3, 2)$, it is not possible to represent more than four messages on the fourth write. Indeed, after the third write, the cell state vector contains five ones and three zeros, so at most $\log_2(3)$ information bits can be conveyed by the remaining zero-valued positions. The message that is stored in

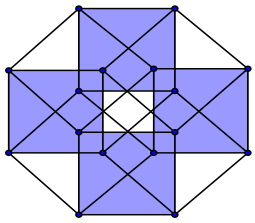

Figure 3.4: $EG(4,2)$, with four parallel planes shaded, as in [50].

the third write can always be represented on the fourth write, simply by leaving the memory state unchanged. Thus, one of at most four messages can be represented on the fourth write.

Example 3.3.2. Using $EG(4,2)$, the resulting WOM code has parameters

$$\langle 16, 16, 16, 12, 8, 8, 8, 4\rangle / \ 16.$$

Recall that $EG(4,2)$, shown in Figure 3.3, has 16 points and 140 planes, and can be partitioned into two parallel 3-flats. The first four writes are the same as in Example 3.3.1, by using the $EG(3,2)$ code on a 3-flat that contains the points corresponding to the first four information messages. After the fourth write, the points in that 3-flat are all programmed to one, and the $EG(3,2)$ WOM code may be applied to the points of the remaining 3-flat to encode the final four writes.

□

Proposition 3.3.3. *The $EG(m,2)$ WOM code achieves $4(m-2)$ writes and has parameters*

$$\underbrace{\langle 2^m, 2^m, 2^m, 2^m - 4, 2^{m-1}, 2^{m-1}, 2^{m-1}, 2^{m-1} - 4, \ldots, 8, 8, 8, 4\rangle}_{4(m-2)} / 2^m.$$

Proof. The cell state vector has length 2^m, equal to the number of points in $EG(m, 2)$. Recall that each cell state vector in the $EG(m, 2)$ WOM code will be Hamming distance one away from a codeword of the Reed-Muller code $\mathcal{R}(m-2, m)$. We proceed by induction on the dimension of the finite geometry. The base case is the $EG(3, 2)$ WOM code. Now suppose that there exists an $EG(k, 2)$ WOM code with the parameters described in the Proposition. Consider the finite Euclidean geometry $EG(k + 1, 2)$, which can be partitioned into two parallel hyperplanes, i.e., two disjoint copies of $EG(k, 2)$. Since any four points lie on a common hyperplane (in fact, many), there exists a hyperplane that contains the points that correspond to the first four information messages to be written. These messages can be encoded using the $EG(3, 2)$ WOM code on a cube within this hyperplane containing those points. After the first four writes, all points in the hyperplane are set to one, and the $EG(k, 2)$ code can be used on the remaining hyperplane. Thus, this $EG(k + 1, 2)$ WOM code allows for $4((k + 1) - 2)$ writes, and has the parameters listed above, with $m = k + 1$.

□

Since codewords of the WOM code are Hamming distance one from a codeword of the corresponding Reed-Muller code, performing majority-logic decoding on a stored cell state vector will provide the location of the position of the "error". The code is designed so that this position corresponds to an information message, i.e., a point in the geometry. Thus, majority-logic decoding identifies the message, and can be used to decode the $EG(m, 2)$ WOM code.

3.3.1 Comparison

Table 3.1 shows the rates of the proposed $EG(m, 2)$ WOM codes and the $PG(m, 2)$ WOM codes from [52] for small values of m. As expected from the geometric struc-

Code	length	rate
PG(2,2)	7	1.60
EG(3,2)	8	1.38
PG(3,2)	15	1.82
EG(4,2)	16	1.66
PG(4,2)	31	1.60
EG(5,2)	32	1.50

Table 3.1: Comparison of rates of small dimension projective and Euclidean geometry WOM codes.

ture, the efficiency of the EG WOM codes is less than that of the PG codes. Indeed, for the special cases when $m = 2$ and 3, the $PG(m, 2)$ WOM codes attain the maximum number of writes indicated in Proposition 3.2.2 [52] due to the fact that the Hamming code is perfect and certain shortened versions retain maximal. The Merkx construction does not have a general geometric description with explicit parameters for $m > 3$. On the other hand, the $EG(m, 2)$ family of codes has a geometric description for all m. Moreover, the $EG(m, 2)$ construction presented here yields a new family of WOM codes with new blocklengths, decent rate, and simple encoding and decoding algorithms. The blocklengths of the EG codes, all powers of two, make them amenable to code concatenation techniques, and the construction shows that variable information WOM codes can be obtained from incidence structures.

In general, designing efficient WOM codes from incidence structures requires low weight incidence vectors, and intersections of these structures that can point to specific messages. In the case of $EG(m, 2)$, the $(m-2)^{th}$ order Reed-Muller code was chosen so that the corresponding minimum weight codewords would be planes and therefore have low weight. Since any two distinct planes intersect in 0 or exactly 2 points, taking unions of multiple planes does not uniquely designate any one partic-

ular point when multiplicity is considered.

Remark 3.3.4. Since the sum-rate of the EG WOM code family decreases as m grows, we compared the strategy of reusing $EG(3,2)$ repeatedly on adjacent sections of the memory to the construction above. The general EG construction outperforms the repeated use of the $EG(3,2)$ code for $m = 4$ and 5, but when $m = 6$ the reapplication of the $EG(3,2)$ code achieves a better sum-rate on the same number of cells, 2^6. However, the number of writes differs—in the case of the $EG(6,2)$ code, 16 writes are achieved, but the reapplication of the $EG(3,2)$ code achieves only four writes. The strategy can be tailored to the needs of the application: many writes but a lower sum-rate, or fewer writes and higher sum-rate.

3.4 Ternary flash codes from $EG(m,3)$

Consider the finite Euclidean geometry of dimension m over $\mathbb{F}_3$, denoted $EG(m,3)$. This incidence structure consists of 3^m points and $\frac{3^m(3^m-1)}{6}$ lines. Each line contains three points, and every point lies on $\frac{3^m-1}{2}$ lines. Given two points $\mathbf{a}, \mathbf{b}$, there is a unique line that contains these points, and a unique third point $\mathbf{c}$ that lies on that line. For this code, we assume each cell has $q = 3$ levels, which we will denote with symbols $0, 1$, and 2, and say that these correspond to increasing cell levels in the memory. Thus $0 < 1 < 2$, even though finite fields do not have a linear ordering. For $\mathbf{x}, \mathbf{y} \in \mathbb{F}_3^m$, the vector notation $\mathbf{x} < \mathbf{y}$ means that $x_i \leq y_i$, for $1 \leq i \leq m$.

3.4.1 Encoding and decoding

We construct a $\langle 3^m \rangle_3^2/(2m)$ WOM code from $EG(m,3)$. Each of the 3^m messages is represented by a point in $\mathbb{F}_3^m$. The memory state vector $\mathbf{c}$ will have $2m$ cells.

For convenience, we organize the memory state vector in the form $\mathbf{c} = (\mathbf{a}, \mathbf{b})$ where $\mathbf{a}, \mathbf{b} \in \mathbb{F}_3^m$. Assume the memory cells are each initialized at 0, i.e., the current memory state vector is $\mathbf{c} = (\mathbf{0}, \mathbf{0})$.

The encoding rule is as follows.

1) First Write: Given message $\mathbf{v} = (v_0, \ldots, v_{m-1}) \in \mathbb{F}_3^m$, if the largest component of $\mathbf{v}$ is at most one, then set $\mathbf{c} = (\mathbf{v}, \mathbf{0})$. If $\mathbf{v}$ contains 2 as an entry, locate the unique line that contains $\mathbf{v}$ and the point $\mathbf{0} = (0, 0)$. There is a unique third point on that line, which we denote by $\mathbf{y}$. If $\mathbf{y}$ has largest component 1, set $\mathbf{c} = (\mathbf{0}, \mathbf{y})$. If $\mathbf{y}$ also contains an entry 2, then choose two points $\mathbf{a}, \mathbf{b}$ that form a line with $\mathbf{v}$ where each has largest component 1, and set $\mathbf{c} = (\mathbf{a}, \mathbf{b})$.

2) Subsequent Writes: Let $\mathbf{c} = (\mathbf{a}, \mathbf{b})$ be the current memory state vector, and suppose the message $\mathbf{v}' = (v'_0, v'_1, \ldots, v'_{m-1})$ is to be stored.

   - If $\mathbf{b} = \mathbf{0}$: If $\mathbf{a} < \mathbf{v}'$, set $\mathbf{c} = (\mathbf{v}', \mathbf{0})$. If $\mathbf{a} \not< \mathbf{v}'$, set $\mathbf{c} = (\mathbf{a}, \mathbf{b}')$ where $\mathbf{b}'$ is the third point on the unique line containing $\mathbf{v}'$ and $\mathbf{a}$.
   - If $\mathbf{a} = \mathbf{0}$ and $\mathbf{b} \neq \mathbf{0}$: Consider the vector $\mathbf{w} = (w_0, w_1, \ldots, w_{m-1})$ where $w_i + v_i \equiv 0 \pmod 3$ for $i = 0, 1, \ldots, m-1$. If $\mathbf{b} < \mathbf{w}$, then set $\mathbf{c} = (\mathbf{0}, \mathbf{w})$. If $\mathbf{b} \not< \mathbf{w}$, then set $\mathbf{c} = (\mathbf{a}', \mathbf{b})$ where $\mathbf{a}'$ is the third point on the unique line containing $\mathbf{v}'$ and $\mathbf{b}$.
   - If $\mathbf{a} \neq \mathbf{0}$ and $\mathbf{b} \neq \mathbf{0}$: Let $\mathbf{y}$ be the third point on the unique line containing $\mathbf{v}'$ and $\mathbf{a}$, and let $\mathbf{x}$ be the third point on the unique line containing $\mathbf{v}'$ and $\mathbf{b}$. If $\mathbf{b} < \mathbf{y}$ and $\mathbf{a} \not< \mathbf{x}$, set $\mathbf{c} = (\mathbf{a}, \mathbf{y})$. If $\mathbf{a} < \mathbf{x}$ and $\mathbf{b} \not< \mathbf{y}$ set $\mathbf{c} = (\mathbf{x}, \mathbf{b})$. If both $\mathbf{b} < \mathbf{y}$ and $\mathbf{a} < \mathbf{x}$, then choose a vector in $\{(\mathbf{a}, \mathbf{y}), (\mathbf{x}, \mathbf{b})\}$ that results in fewer cell increases.

Figure 3.5: $EG(1,3)$

If none of the above, then consider the $\frac{3^m-1}{2}-2$ lines incident with $\mathbf{v}'$ that do not contain $\mathbf{a}$ or $\mathbf{b}$. Suppose the i^{th} line consists of points $\{\mathbf{v}', \mathbf{w^i}, \mathbf{z^i}\}$ for $i = 1, \ldots, \left(\frac{3^m-1}{2}-2\right)$. Let $\mathcal{J} = \{i|(\mathbf{a},\mathbf{b}) < (\mathbf{w^i},\mathbf{z^i})\}$. If $\mathcal{J} \neq \emptyset$ then set $\mathbf{c} = (\mathbf{w^i},\mathbf{z^i})$ for some $i \in \mathcal{J}$ for which the vector $(\mathbf{w^i},\mathbf{z^i})$ has minimum weight. If such a vector does not exist, then the message $\mathbf{v}'$ can not be written.

Two writes are guaranteed because the weight of the memory state vector is either one or two after the first write.

The rule for decoding is as follows.

- If the memory state vector is

$$\mathbf{c} = (a_0, a_1, \ldots, a_{m-1}, b_0, b_1, \ldots, b_{m-1}),$$

  then when $b_0 = b_1 = \cdots = b_{m-1} = 0$, decode to the point $(a_0, a_1, \ldots, a_{m-1})$.

- Otherwise, $\mathbf{c}$ decode to the point $(v_0, v_1, \ldots, v_{m-1})$ such that $v_i + a_i + b_i \equiv 0 \pmod 3$ for $i = 0, 1, \ldots, m-1$.

3.4.2 Examples

Example 3.4.1. Consider $EG(1,3)$, consisting of one line and three points, as shown in Figure 3.5. The corresponding WOM code has parameters $\langle 3 \rangle_3^2/2$ and rate $\frac{2\log_2(3)}{2}$. The code based on $EG(1,3)$ has a simple encoding map, shown by the tree in Fig-

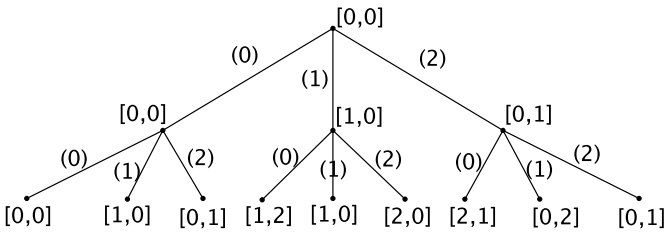

Figure 3.6: $EG(1,3)$ WOM code, where the i^{th} layer of the tree corresponds to the i^{th} write.

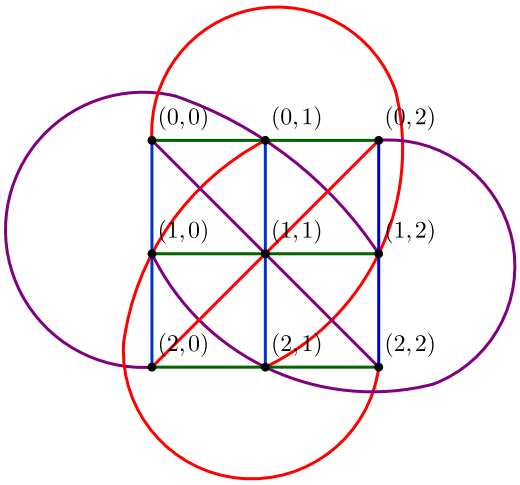

Figure 3.7: The finite geometry $EG(2,3)$, with color classes denoting bundles of parallel lines.

ure 3.6. Note that more than two writes are possible in some cases, but we only demonstrate the guaranteed writes in the figure.

□

Example 3.4.2. Consider $EG(2,3)$, consisting of nine points and 12 lines in Figure 3.7. We construct a $\langle 9 \rangle_3^2/4$ WOM code from $EG(2,3)$. Each of the 9 messages is represented by a point. The memory state vector $\mathbf{c}$ will have four cells, denoted

$(\mathbf{a}, \mathbf{b})$ where $\mathbf{a}, \mathbf{b} \in \mathbb{F}_3^2$. While the code guarantees two writes, often more are possible.

The following is an example message sequence that can obtain five writes:

Info.	(01)	(22)	(21)	(00)	(02)
Writes	$[0, 1, 0, 0]$	$[0, 1, 1, 0]$	$[0, 2, 1, 0]$	$[1, 2, 2, 1]$	$[1, 2, 2, 2]$

In contrast, the message sequence below yields two writes:

Info.	(12)	(20)
Writes	$[1, 0, 1, 1]$	$[2, 1, 2, 2]$

When viewed as a variable-rate WOM code, the $EG(2, 3)$ code obtains 3.108 writes, on average. This average was obtained by running 10^6 random message sequences in MATLAB and averaging over the number of writes achieved. If we restrict certain bad message sequences such as that above, more writes may be guaranteed, but at a significant cost in the number of messages that can be represented at each generation. Consider the trivial scheme of representing some nonzero message $\mathbf{v}$ on the first write using $(\mathbf{v}, \mathbf{0})$, and a nonzero message $\mathbf{v}'$ on the second write using $(\mathbf{v}, \mathbf{v}')$. The $EG(2, 3)$ construction only does better than this scheme when it attains three or more writes. □

Example 3.4.3. Consider $EG(3, 3)$, consisting of 27 points and 117 lines, shown partly in Figure 3.8. The corresponding WOM code has parameters $\langle 27 \rangle_3^2/6$ and rate $\frac{2 \log_2(27)}{6}$. □

Remark 3.4.4. We observe that attempting to create flash coding schemes with finite geometries over alphabets with $q > 3$ generally does not yield codes that are more efficient for multi-level flash memory. The problem is that many points in the

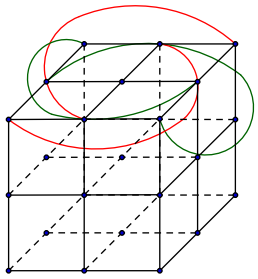

Figure 3.8: $EG(3,3)$, with select lines drawn.

geometry have labels that would require writing $q-1$ in a cell during the first write, which effectively prevents that cell from being reused (until the erase operation is performed). However, the incidence structure of finite geometries over higher alphabets remain a good source for constructions of *binary* WOM codes. In the following example, we use $EG(2,3)$ to create a binary WOM code of length nine with sum rate 1.41.

Example 3.4.5. Figure 3.7 shows $EG(2,3)$, which we will use to create a binary WOM code of length nine. Since every line has three points and each pair of points is contained in a unique line, we can create an encoding map similar to the process in described in Sections 3.2 and 3.3. We construct a $\langle 9,9,9,9\rangle^4/9$ WOM code. The length-nine cell state vector will be an indicator vector of the points, labeled $\{1,2,\ldots,9\}$. The four writes are as follows:

1. To store the point i on the first write, place a one in the i^{th} position in the vector.

2. To store the point $j \neq i$ on the second write, find the unique line that contains i and j. There is a unique third point on that line, k. Place a one in the k^{th} position in the cell state vector.

3. The third write is characterized by a line union a point. If the message is i, then choose any line L that contains k (with $L \neq \{i, j, k\}$), and place ones in the positions indicated by the points in L (follow a similar process if the message is k). If the message is $l \neq i, k$, then place ones in positions j and l of the vector.

4. On the final write, the message is indicated as either the intersection of two lines, or it is the point corresponding to the only position in the vector that has a zero in it.

□

One of the advantages of the $EG(m, q)$ families of codes is that they are good candidates for concatenation-type schemes. Their simple encoding and decoding maps allow for repeated use of the codes as components of larger schemes without hampering the efficiency of the encoding and decoding of the overall code. In the next chapter, we will use the ternary codes as component codes for a scheme called *generalized coset encoding*, and the binary Euclidean geometry WOM codes will be used as components in multi-level concatenation.

Chapter 4

Coding methods for multi-level flash memories[1]

The development of flash memory cells on $q > 2$ levels has renewed interest in efficient coding strategies for 'generalized' write-once memories, i.e., those with greater than two states per cell. This chapter is devoted to approaches to designing flash codes from WOM codes. In Section 4.1, we present strategies for the efficient reapplication of WOM codes to q-ary cells. Section 4.2 discusses methods of concatenating WOM codes and error-correcting codes that result in a variety of flash coding schemes. In Section 4.3 we present a construction called *generalized position modulation*, which uses a component flash code to create a longer code with greater rewriting capability. Finally, Section 4.4 presents a generalization of the classical coset encoding scheme. The original construction uses the cosets of an error-correcting code to create a WOM code; the generalization presented in this thesis details a method for using component flash codes in order to apply coset encoding when $q > 2$. Together, these approaches

[1]Material in this chapter first appeared in [23], the *Proceedings of the Int'l Castle Meeting on Coding Theory and Applications* (Sections 4.1, 4.2), and in [24], the *Proceedings of the Asilomar Conference on Signals, Systems, and Computing* (Section 4.4).

x	$\mu^1(x)$	$\mu^2(x)$	$\mu^3(x)$	$\mu^4(x)$
00	000	111	111	222
01	100	011	211	122
10	010	101	121	212
11	001	110	112	221

Table 4.1: Rivest-Shamir code adapted to $q = 3$ levels.

yield codes with a wide variety parameters that can be applied in any asymmetric memory setting.

4.1 Strategies for reusing binary WOM codes

A natural approach to creating flash codes is to reuse WOM codes on q-ary cells. The strategies presented here make use of efficient existing codes and also provide a basis for comparison for new flash coding schemes. In this section we examine construction methods for adapting binary WOM codes for use on multi-level cells.

One way to use binary codes[2] on q-level cells is to read the cells modulo 2. A naive approach is to let the set of codewords consist of all cell state vectors that reduce modulo 2 to a binary codeword. A more efficient application of a $\langle v \rangle^t / n$ code to q-level cells is to increase the charge of all cells to 1 after the t^{th} write, and then employ the code again. We will refer to this scheme as the *complement scheme*, since reduction modulo 2 either reveals a WOM codeword or the complement of a codeword. More precisely, in the complement scheme, let x denote the information message, and $c^i(x)$ be a codeword that represents x on the i^{th} write. We reuse the binary WOM code by taking $c^{t+i}(x) = c^i(x) + \mathbf{1}$, for $i < t$, where $\mathbf{1}$ is the all ones vector. Similarly, after

[2] The idea of reducing the cell state vectors modulo 2 was also used in [28] to adapt *classical* codes for use on multi-level cells.

mt writes, the cell values are increased to m, and we set $c^{mt+k}(x) = c^k(x) + m \cdot \mathbf{1}$ for $k = 1, \ldots, t-1$. Note that this scheme guarantees $(q-1)t$ writes. Table 4.1 shows Example 2.2.2 adapted to $q = 3$-level cells in this way.

We will use this simple scheme as a basis for comparison when considering the following methods of adapting binary WOM codes to q levels.

Construction: Consider a $\langle 2^k \rangle^t / n$ WOM code. Let x be a binary information sequence of length k, and let $U(x) = \{u : u = c^i(x) \text{ for some } i = 1, \ldots, t\}$. Let s be a length n cell state vector representing the message x. Given s, suppose we want to write a new message $y \neq x$. Let V be the set of n-tuples with all entries even (possibly 0) and less than q. We present two strategies.

- **Strategy A**: To minimize the number of cells that are increased, search the set $U(y) + V$ for the representation whose difference from s requires the fewest cells to increase. Thus, look for $s' \in U(y) + V$ such that $s' \geq s$ (componentwise, all entries in s' are at least as much as those in s) and further that s and s' differ in the least number of places, i.e. the Hamming weight, $wt_H(s' - s)$ is minimized. The new cell state vector is s' and represents the new message y. In searching the set $U(y) + V$ as the cell values approach q, we omit the values of s' that would cause a block erasure.

- **Strategy B**: To minimize the magnitude of the resulting cell state vector s', search the set $U(y) + V$ for the representation whose difference from s is such that the maximum cell entry of s' is minimized. If there is a tie, arbitrarily choose one that requires the fewest number of cells to increase. Thus, look for $s' \in U(y) + V$ such that $s' \geq s$ and that the maximum entry in s' is the smallest.

For specific codes, the strategies can be described more explicitly. For example, the following flash code encoding map is based on Example 2.2.2, and uses reduction modulo 2 to identify the decoding map from the cell state vectors to the variable vectors. Following Strategy A, the rewriting rule is as follows. Let s be the current cell state vector representing the message x, and y the new message to be written.

- If $x, y \in \mathbb{F}_2^2 \setminus \{00\}$,
  - If $s \bmod 2 = c^1(x)$, add the weight one vector $w = c^2(y) - c^1(x)$ to the current state, to obtain the new cell state vector $s' = s + w$.
  - If $s \bmod 2 = c^2(x)$ write $w = c^1(z)$, where $z \in \mathbb{F}_2^2 \setminus \{00, x, y\}$, to obtain $s' = s + w$.
- If $x = 00$, write $c^1(y)$.
- If $y = 00$, then if $s \bmod 2 = c^1(x)$, add $c^1(x)$ to s; otherwise add $\mathbf{1} - c^2(x)$ to s.

Following Strategy B, the rewriting rule depends on the actual magnitude (in $\{0, \ldots, q-1\}$) of each cell entry.

The general rule is to increase a subset of the cells such that the new vector reduces to either $c^1(y)$ or $c^2(y)$ modulo 2 and no one cell is allowed to gain too much charge.

Example 4.1.1. Using the rules above for the Rivest-Shamir WOM code in Example 2.2.2, suppose the following information sequence is to be stored in a given set of cells with $q = 4$ levels.

$$11 \to 00 \to 01 \to 10 \to 11 \to 01$$

Following Strategy A, the sequence of cell state vectors is as follows

$$A : 001 \to 002 \to 102 \to 103 \to 203 \to 213$$

Following Strategy B, the sequence of cell state vectors is as follows

$$B : 001 \to 111 \to 211 \to 212 \to 312 \to 322$$

□

Example 4.1.2. To further illustrate the different strategies, consider writing the sequence $1 \to 2 \to 1 \to 3$ using the $PG(2,2)$ WOM code in Example 3.2.1, where the labeling on the Fano Plane is as in Figure 1. Following Strategies A and B, the sequence of cell state vectors is as follows:

$$A : (1000000) \to (1001000) \to (1002000) \to (1002001)$$

$$B : (1000000) \to (1001000) \to (1001101) \to (1101111)$$

□

4.1.1 Analysis of Strategies A and B

The expected number of writes for floating codes was studied in [15, 6] and can be more important than the worst case analysis in determining which codes to use in practice. Code constructions in [30] have a guarantee of $(q-1) + \lfloor \frac{q-1}{2} \rfloor$ writes for a $k = 2$-dimensional message space and $n = 2$ cells. The same paper also proved the existence of floating codes that achieve $(q-1)n - o(n)$ writes as $n \to \infty$ for fixed k and q. Asymptotically optimal codes for the average case with $k = 2$ have been constructed where the expected number of writes grows like $n(q-1) - o(q)$ [6]. Both cases include the assumption that only one cell level changes at each write, which is reasonable when $n \gg 2^k$. However, since Strategies A and B are intended to be

used for *any* WOM code, not just those that meet this criterion, we do not use this assumption.

The guaranteed number of writes using Strategy B for the $\langle 4\rangle^2/3$ Rivest-Shamir WOM code on q level cells is $2(q-1)$. This can be seen by examining a sequence of messages that causes the maximum number of cell increases under Strategy B. For example, the alternating sequence of messages $00 \to 01 \to 00 \to 01 \to 00 \to \cdots \to 01 \to 00$ has cell state vector sequence $000 \to 100 \to 111 \to 211 \to 222 \to \cdots \to (q-1)(q-2)(q-2) \to (q-1)(q-1)(q-1)$. Observe that for every two writes, the cell state vector does not increase a cell level more than once, and both representations of a given message are used. Thus, the guaranteed number of writes using Strategy B is $2(q-1)$.

The guaranteed number of writes using Strategy A for the $\langle 4\rangle^2/3$ Rivest-Shamir WOM code on q level cells is also $2(q-1)$. Again we consider a sequence of messages that causes the maximum number of cell increases. For example, the alternating sequence of messages $00 \to 01 \to 00 \to 01 \to 00 \to \cdots \to 01 \to 00 \to 01$ has cell state vector sequence $000 \to 100 \to 200 \to 300 \to 400 \to \cdots \to (q-2)00 \to (q-1)00 \to (q-1)11 \to (q-1)22 \to \cdots \to (q-1)(q-1)(q-1)$. Observe that the first $q-1$ writes follow the Strategy A protocol to increase the fewest number of cells, but that once any cell attains the maximum charge, the strategy continues to write using the next best representation choice for each message. Thus, a total of $2(q-1)$ writes are guaranteed.

The following theorem shows that the guaranteed number of writes for both Strategies A and B is at least as good as the complement scheme for any general binary WOM code.

Theorem 4.1.3. *Let $\mathcal{C}$ be a $\langle v\rangle^t/n$ binary WOM code. Then the guaranteed number*

of writes by applying either Strategy A or Strategy B to $\mathcal{C}$ on q-level flash cells is at least $(q-1)t$.

Proof. We proceed by induction on q. For $q = 2$, the WOM code already guarantees t writes. So assume the hypothesis holds for $q = r$. That is, for any sequence of messages, we are guaranteed at least $(r-1)t$ writes using Strategy A or B. Now let us consider the case when $q = r + 1$. Then for any sequence of $(r-1)t$ messages, using Strategy A or Strategy B, by the induction hypothesis we will reach a cell state vector $(c_1, c_2, \ldots, c_n)$, with entries $c_i \leq r-1$, $i = 1, 2, \ldots, n$. We can now artificially increase each cell levels to $r-1$ at the end of $(r-1)t$ writes to yield a cell state vector $(r-1, r-1, \ldots, r-1)$. Without loss of generality, the cell state vector $(r-1, r-1, \cdots, r-1)$ can be thought of as being the all-zero vector $(0, 0, \cdots, 0)$. It is now easy to see that either Strategy A or Strategy B will allow us to write at least t more times using the original t writes of the binary WOM code $\mathcal{C}$. Thus, a total of rt writes is guaranteed for either strategy when $q = r + 1$, thereby proving the result. □

To see if the lower bound of $(q-1)t$ writes is met in Theorem 4.1.3, the weight distributions of the different representations for each message in the original WOM code have to be taken into account. For example, for two-write WOM codes where the minimal weight representation for each message is unique, the guaranteed number of writes is $2(q-1)$ as above.

Strategies A and B applied to the Rivest-Shamir code each guarantee two writes when $q = 2$ and four writes when $q = 3$, whereas the expected number of writes using the strategies for this code (assuming a uniform distribution on the message space) is approximately 2.47 for $q = 2$ and 4.89 for $q = 3$ for each case. Note that the simple application of the Rivest-Shamir code to q-level cells using the complement

scheme requires $q \geq 3$ to get more than two guaranteed writes. Figure 4.1 compares the average number of writes of the complement scheme, Strategy A, and Strategy B on q-level cells when applied to the binary Rivest-Shamir WOM code from Example 2.2.2. In Monte Carlo simulations, 10^5 random message sequences were generated and the number of writes was recorded for the three different methods. As shown in Figure 4.1, the strategies applied to the Rivest-Shamir code exhibit a noticeable gain over the the complement scheme that is growing as $q \to \infty$. However, the average number of writes for each strategy is still quite far from the capacity limit on the number of writes possible for representing four messages per write using three cells on q levels (see Theorem 2.2.4).

Strategies A and B did not exhibit much gain over the complement scheme when the $PG(2, 2)$ code in Example 3.2.1 was simulated for small q. This is possibly due to the near-optimality of the $PG(2, 2)$ WOM code. Further, it is likely that in general, the more optimal a code is, the less it will benefit from the strategies, since the reapplication of the code under the complement scheme already generates an efficient code.

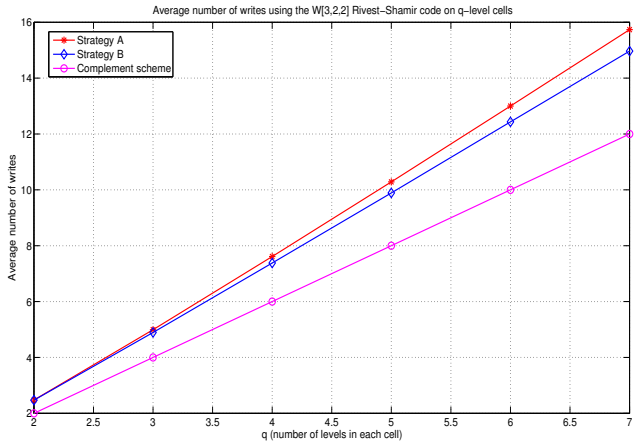

Figure 4.1: Comparison of the average number of writes achieved by Strategies A and B and the complement scheme.

In [6], two coding schemes are presented that have a similar flavor to Strategies A and B, but apply in the different setting of random floating codes. In that work, the authors propose two random coding schemes: a "Simple scheme" that randomly chooses to increase a single cell by one, and a "Least scheme" that chooses a message representation that increases the coordinate with the lowest charge level. In contrast, Strategies A and B in this dissertation apply to any WOM code without the assumption that only one cell increases at each write.

4.2 Concatenation with WOM codes

In this section we consider ways that code concatenation may be used to obtain new WOM or flash codes. Let $[n, k, d]_q$ denote a classical q-ary linear code of block length n, dimension k, and minimum distance d. Two classical codes may be concatenated as follows.

Definition 4.2.1. Let $\mathcal{A}$ be an $[n_1, k_1, d_1]_{q^{k_2}}$ code and $\mathcal{B}$ be an $[n_2, k_2, d_2]_q$ code. Then

the concatenated code $\mathcal{C} = \mathcal{A} \boxtimes \mathcal{B}$ is an $[n_1 n_2, k_1 k_2, d_1 d_2]_q$ code with outer code $\mathcal{A}$ and inner code $\mathcal{B}$. The k_1 information symbols (each chosen from a q^{k_2}-ary alphabet) are first encoded into n_1 symbols using $\mathcal{A}$. Each of the encoded symbols is then represented by k_2 q-ary symbols. Each group of these k_2 symbols is then encoded into n_2 q-ary symbols using $\mathcal{B}$. Thus, $n_1 n_2$ encoded symbols are obtained to form a codeword in $\mathcal{C}$.

The above concatenation may be seen by the following mapping

$$\mathbb{F}_{q^{k_2}}^{k_1} \xrightarrow{\mathcal{A}} \mathbb{F}_{q^{k_2}}^{n_1} \xrightarrow{q\text{-ary representation}} \mathbb{F}_q^{n_1 k_2} \xrightarrow{\mathcal{B}} \mathbb{F}_q^{n_1 n_2}$$

Concatenating classical codes with binary WOM or flash codes yields codes with both error correction and rewrite capabilities.

Several researchers have observed that an outer $\langle 2^k \rangle^t / n$ WOM code $\mathcal{A}$ when concatenated with an inner $[m, 1]_2$ repetition code $\mathcal{B}$ yields a $\langle 2^k \rangle^t / nm$ binary WOM code $\mathcal{C} = \mathcal{A} \boxtimes \mathcal{B}$, where C can correct $\lfloor \frac{m-1}{2} \rfloor$ errors [84, 81, 29]. We expand on these ideas to obtain codes for multi-level flash cells.

A code $C_W \boxtimes C_R$, where C_W is a WOM code and C_R is a length-m repetition code, can be employed as an error-correcting code on q-level cells with the following strategy: on the first write, the binary codeword is written on the cells. An error can be detected by majority decision among each set of m consecutive positions. For subsequent writes and error correction, we will read the q-ary vector as a binary codeword from C_W, by reducing the values in the cells modulo 2. In particular, if a one was erroneously written on the first write in a cell that should have contained a zero, we correct the error by increasing the level of the cell to 2, which is viewed as a 0 (modulo 2). The error has been corrected in the binary word that is read, and the code can correct $\lfloor \frac{m-1}{2} \rfloor$ errors on each write. Subsequent writes are achieved by

increasing chosen cell levels to obtain the desired parity, modulo 2.

The following theorem uses this method to obtain an error-correcting WOM code. Note that errors can occur in either direction and are assumed to be of magnitude one.

Theorem 4.2.2. *Let C_W be a $\langle 2^k \rangle^t / n$ WOM code and let C_R be the $[m, 1, m]_2$ repetition code. The code $C_W \boxtimes C_R$ is an $\langle 2^k \rangle^t / mn$ $\lfloor \frac{m-1}{2} \rfloor$-error-correcting WOM-code on SLCs. Moreover, applied to q-level cells and using the reduced binary vector representation, $C_W \boxtimes C_R$ is a $\langle 2^k \rangle_q^{t'} / mn$ flash code, where $t' = \lceil \frac{(q-1)t}{3} \rceil$ and $\lfloor \frac{m-1}{2} \rfloor$ errors can be corrected at each write.*

Proof. For $q = 2$ the resulting code is a $\langle 2^k \rangle^t / mn$ $\lfloor \frac{m-1}{2} \rfloor$-error correcting WOM code. For any q, the length mn-code has dimension k. We show that the worst-case number of rewrites is $\lceil \frac{(q-1)t}{3} \rceil$. The code $C_W \boxtimes C_R$ is still binary, but we use it on the q-ary cells by reading the information stored in the cells via the reduced binary vectors. Up to $\lfloor \frac{m-1}{2} \rfloor$ errors can be detected and corrected at each write. Error correction consists of increasing the charge level of the cell by one to correct the parity in that entry of the reduced binary vector. In the worst case, an error occurs in the same position on every write, and so that position sees an increase of three levels at each write. However, in the absence of errors we could achieve $(q-1)t$ writes due to the rewriting capability of C_W and the reapplication of the WOM code on q-level cells. Thus, the worst-case number of writes in the error case is $\lceil \frac{(q-1)t}{3} \rceil$. □

As an example of the reading process, if $q = 4, n = 1, m = 3$, the sequence (332) in a cell-state vector would be read as (110) in $C_W \boxtimes C_R$, and decoded to (111) using majority rule. As an example of the error-correction process, consider a cell that is meant to be increased to 0 (modulo 2); if an error causes the cell to instead be read as 1 (modulo 2), then to correct it the charge is increased again. Thus that cell has

seen a total increase of three levels on that write cycle. A similar idea of increasing the cell levels to correct for errors has also been considered in [29, 35].

Example 4.2.3. Let C_W be the $\langle 4\rangle^2/3$ WOM code defined in Example 2.2.2 and let C_R be the $[3,1,3]_2$ repetition code. Then the code $C_W \boxtimes C_R$ is a $\langle 4\rangle^2/9$ single error-correcting WOM code on SLCs (first observed in [84]). Moreover, on q-level cells, the code $C_W \boxtimes C_R$ is a $\langle 4\rangle_q^{\lceil \frac{2(q-1)}{3} \rceil}/9$ single error-correcting flash code. □

Example 4.2.4. Let C_W be the $\langle 7\rangle^4/7$ code based on $PG(2,2)$ from [52] and let C_R be the $[3,1,3]_2$ binary repetition code. Then the code $C_W \boxtimes C_R$ is a $\langle 7\rangle^4/21$ single error-correcting WOM code on SLCs. Moreover, on q-level cells, the code $C_W \boxtimes C_R$ is a $\langle 7\rangle_q^{\lceil \frac{4(q-1)}{3} \rceil}/21$ single error-correcting flash code. □

We next show how to obtain a flash code with increased error-correction by concatenating an inner flash code with an outer classical code.

Theorem 4.2.5. *Let C_1 be an $[n_1,k_1]_{q^{k_2}}$ code that corrects e errors, and C_2 a $\langle 2^{k_2}\rangle_q^t/n_2$ E-error-correcting WOM code. Then $C_1 \boxtimes C_2$ is a $\langle 2^{k_1 k_2}\rangle_q^t/(n_1 n_2)$ WOM code capable of correcting $(\mathbf{E}+\mathbf{1})(\mathbf{e}+\mathbf{1})-\mathbf{1}$ errors.*

Proof. The length and dimension of $C_1 \boxtimes C_2$ is immediate. This code achieves t writes since the inner flash code is capable of t writes. The minimum number of errors that must occur for a decoding failure is $(E+1)(e+1)$, where $E+1$ errors occur among each of $e+1$ distinct length-k_2 q-ary expansions of symbols in C_1. Any smaller number of errors can be corrected by the length $n_1 n_2$ concatenated code. □

For comparison, we show the concatenation of a inner binary repetition code with a classical binary outer code for use on q-level flash cells.

Theorem 4.2.6. *Let C be an $[n,k,d]_2$ e-error-correcting code and let C_R be the $[2m+1,1,2m+1]_2$ binary repetition code. Then the code $C \boxtimes C_R$ for q-level cells results*

in a $\langle 2^k \rangle_q^t / ((2m+1)n)$ *flash code that corrects* $(me + m + e)$ *errors and guarantees* $t = \lceil \frac{q-1}{3} \rceil$ *writes.*

Proof. The length and dimension follow from the construction. Concatenating two binary codes results in a binary code, but we use reduction modulo 2 to adapt the code to q-ary cells. Errors that result in a change in parity of a cell can be corrected by increasing the level of the cell by one. In the worst case, an error occurs in the same cell at every write. In order to correct it, the cell level is increased by one so that it has the same parity as the entry before the error occurred. Thus this code guarantees $\lceil \frac{q-1}{3} \rceil$ writes. Note that the outer code can correct up to e errors and the inner code can correct up to m errors. Thus, the concatenated code can tolerate $(m+1)(e+1) - 1 = me + m + e$ errors. □

Observe that this use of a classical code on multi-level cells gives better error-correction capabilities than the code in Theorem 4.2.2 but can tolerate fewer rewrites since the only rewrite capabilites come from the number of levels.

Example 4.2.7. Let C be an $[n, k, d]_2$ e-error-correcting code and let C_R be the $[3, 1, 3]$ binary repetition code. Then the code $C \boxtimes C_R$ for q-level cells yields a $\langle 2^k \rangle_q^t / (3n)$ flash code that corrects $\mathbf{2e + 1}$ errors and gets $t = \lceil \frac{q-1}{3} \rceil$ writes. □

4.3 Generalized position modulation

In 2009, Wu and Jiang proposed a WOM code construction called *position modulation* [76]. The idea is to partition the block of cells in the memory into sub-blocks, each of equal size, and use the position and contents of the non-zero sub-blocks to convey information. In that paper, the authors showed that taking sub-blocks of size two can yield a WOM code that achieves half the optimal rate for a fixed number of writes.

Low encoding and decoding complexity is a key feature of the codes, which relies on a polynomial-time-implementable mapping from the integers $\{0, \ldots \binom{n}{k} - 1\}$ to a binary vector of length n containing k ones.

More specifically, a block of n cells is partitioned into m sub-blocks of size k, and depending on the amount of information to be stored at each write, j_1 of the sub-blocks are chosen to be made non-zero on the first write. Each of these sub-blocks can contain one of $2^k - 2$ messages. The positions of the j_1 non-zero sub-blocks and their contents encode the information. Before the second write, these j_1 sub-blocks are all entirely programmed with ones (if necessary, additional sub-blocks are also programmed to the all-ones vector before the second write). Then the process is repeated. Given an initial number of writes t and the desired amount of information to be stored at each write, the code length and sub-block size is chosen so that the corresponding position modulation code achieves these goals.

Here we present a *generalized position modulation* (GPM) scheme that uses a component t-write WOM code to create a new WOM code with increased rewrite capabilities. One special case of a GPM construction yields $2t$ writes, while the general construction can achieve more than $2t$ writes. We also describe how GPM codes compare to the original position modulation codes in [76], as well as other existing WOM codes.

We construct a WOM code on $n = hm$ cells, using a component t-write WOM code of length m on each of the h sub-blocks. We will call a group of m cells (a particular sub-block) *active* if there is at least one nonzero cell in the group. A group that is composed of m cells with maximum charge $q-1$ will be called *saturated*, and a group with all zeros is called *empty*. The cell state vector begins with h empty groups. On the first write, k_1 empty groups are chosen and activated, using the component WOM code. The positions of the activated groups and their contents both convey

information. On the second write, the groups activated in write one can be rewritten using the component WOM code, and consequently will contain second generation words from the code. Simultaneously, a new collection of empty groups (sub-blocks) are chosen and activated with first-generation WOM codewords, as in Figure 4.2. The process continues until t writes have occurred, at which point the groups that were activated on the first write are set to the saturated state, as in Figure 4.3. In the following section we present a method for obtaining GPM codes with at least twice the rewrites as the component code.

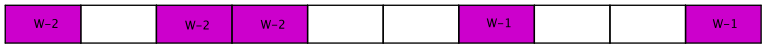

Figure 4.2: A GPM cell state vector, split into h groups of m cells, where $w - i$ denotes an i^{th} generation word in the component code.

Figure 4.3: Once an active component exhausts its t writes, all m cells are set to 1, shown by the darker shading.

4.3.1 GPM-WOM code construction

Given a $\langle v \rangle^t / m$ WOM code $\mathcal{C}$, we will construct a $\langle v_1, \ldots, v_T \rangle / hm$ WOM code $\mathcal{C}'$. Start by partitioning the cells into h groups, with m cells in each group. For $i = 1, \ldots, T$ let K_i denote the set of new groups chosen in the i^{th} write, and let $k_i = |K_i|$. For $i \leq t$, the groups that are active during the i^{th} write are $K_1 \cup \cdots \cup K_i$. For $t < i \leq T$, the groups that are active during the i^{th} write are $K_i \cup K_{i-1} \cup \cdots \cup K_{i-t+1}$. Let $N_i := k_1 + k_2 + \cdots + k_i$ denote the number of groups that are nonzero at the end of the i^{th} generation. For $i \leq 0$, define $N_i = 0$. Since this scheme does not erase the

cells in K_i before writing information in K_{i+1}, distinguishing the new groups requires that we can distinguish all first-generation WOM codewords in $\mathcal{C}$.

Theorem 4.3.1. *Given $v_1, \ldots, v_T$, $m \in \mathbb{N}$, and a fixed component WOM code $\mathcal{C}$ with parameters $\langle v \rangle^t / m$, then if h and $k_1, \ldots, k_s$ satisfy*

1. $v_1 \leq \sum_{k_1=1}^{h-(s-1)} \binom{h}{k_1} (v-1)^{k_1}$,
2. *For $i \leq s$,* $v_i \leq (v-1)^{N_{i-1}-N_{i-t}} \sum_{k_i=1}^{h-(s-i)-N_{i-1}} \binom{h-N_{i-1}}{k_i} (v-1)^{k_i}$,
3. *For $i \geq s+1$,* $v_i \leq (v-1)^{N_i - N_{i-t}}$,

there exists a $\langle v_1, \ldots, v_T \rangle / (mN_s)$ WOM code.

Proof. On the first write, choose the k_1 groups of cells in K_1 such that $1 \leq k_1 \leq h-t$. Using $\mathcal{C}$, each group can represent one of $v-1$ information symbols (excluding the all zeros vector, since the k_1 chosen groups must be distinguishable from the $h - k_1$ zero groups). Thus, there are v_1 possible states that can be stored on the first write, where

$$v_1 = \sum_{k_1=1}^{h-(s-1)} \binom{h}{k_1} (v-1)^{k_1}.$$

Since $k_i \geq 1$ for all $i = 1, \ldots, s$, it is necessary to restrict the possible value of k_1 to be less than $h - (s-1)$. Now, instead of performing the soft-erase operation detailed in [76] (setting all active groups to the all-ones word), we will use $\mathcal{C}$ to write on the k_1 nonzero groups again in the next $t-1$ writes. It remains necessary, however, to distinguish the groups chosen in the first write from those chosen on future writes. We therefore require that those groups get rewritten as something different (whose generation is distinguishable from previous generations) at each of

the following writes. The soft-erase operation will be performed on these groups after the t^{th} write.

The second write will proceed as follows: a new message must be written on each of the k_1 non-zero groups in K_1 using the WOM code $\mathcal{C}$, and we choose k_2 new groups with $1 \leq k_2 \leq h - k_1 - (s-2)$ from the remaining zero positions. Each group receives one of $v-1$ messages. It is possible to represent up to v_2 messages in the second write, where

$$v_2 = (v-1)^{k_1} \sum_{k_2=1}^{h-k_1-(s-2)} \binom{h-k_1}{k_2} (v-1)^{k_2}.$$

The term $(v-1)^{k_1}$ comes from using $\mathcal{C}$ to rewrite on the groups in K_1. The remaining terms result from chosing the groups to activate on the second write, and also choosing the codewords to write in each of those active locations. Let $N_i := k_1 + k_2 + \cdots + k_i$, for $i \geq 1$ and define $N_j := 0$ for $j \leq 0$.

After the t^{th} write, the k_1 groups in K_1 cannot tolerate any further writes. Thus, at the start of the $t+1^{th}$ write, first perform the soft-erase operation by setting all of the $k_1 m$ cells in K_1 to ones, and continue to write on the remaining $(h-k_1)m$ cells. In general, the soft-erase operation will be applied to the cells in group K_i on the $(i+t)^{th}$ write.

In the i^{th} write where $i \leq s$, the number of possible messages that may be represented is

$$v_i \leq (v-1)^{N_{i-1}-N_{i-t}} \sum_{k_i=1}^{h-(s-i)-N_{i-1}} \binom{h-N_{i-1}}{k_i} (v-1)^{k_i}.$$

The most recently activated groups can always be identified from the generation-one codewords they contain. After a group has been active for t writes, the group is programmed to all ones.

New groups are activated after the t^{th} write, up until the s^{th} write, since $N_s = h$.

	Known WOM code parameters	Known WOM code rate	PM code rate	GPM code rate
t=2	$\langle 26\rangle^2/7$	1.34	1.14	–
t=3	$\langle 63\rangle^3/12$	1.49	1.35	1.46
t=4	$\langle 7\rangle^4/7$	1.60	1.49	1.54
t=5	$\langle 11\rangle^5/11$	1.57	1.63	1.66
t=6	$\langle 16\rangle^6/15$	1.60	1.71	1.73
t=7	$\langle 15\rangle^7/15$	1.82	1.81	1.79
t=8	$\langle 15\rangle^8/19$	1.65	1.88	1.83
t=9	$\langle 15\rangle^9/21$	1.67	1.95	1.87
t=10	$\langle 15\rangle^{10}/24$	1.63	2.01	1.90

Table 4.2: Table of WOM code, position modulation code, and GPM code rates for given values of t.

Therefore when $i \geq s+1$, the number of messages that can be represented on the i^{th} write is $v_i \leq (v-1)^{N_i - N_{i-t}}$.

The result is a $\langle v_1, \ldots, v_T\rangle/hm$ WOM code.

□

Remark 4.3.2. The constructions resulting from Theorem 4.3.1 require that the codewords of $\mathcal{C}$ can be partitioned by write-generation, and that neither the all zeros word nor the all ones word is a codeword.

4.3.2 Examples and code performance

In this subsection we provide two examples of a GPM code using a restricted version of the $\langle 4\rangle^2/3$ Rivest-Shamir code in Example 2.2.2 as the component code, and also a version of the Merkx WOM code in Example 3.2.1 as a component code. In the Rivest-Shamir case, we eliminate the message 00 since the corresponding WOM codewords for that message are 000 and 111, which are prohibited by Theorem 4.3.1. This gives a $\langle 3\rangle^2/3$ WOM code as the component code.

Example 4.3.3. Take $m = 3$ and $h = 50$, and $T = 4$. Then the resulting GPM code satisfies the following constraints, assuming $k_1 = 25, k_2 = 13, k_3 = 12$.

$$\begin{aligned} v_1 &= \binom{50}{25} 3^{25}, \\ v_2 &= \binom{25}{13} 3^{25+13}, \\ v_3 &= 3^{13+12}, \\ v_4 &= 3^{12}. \end{aligned}$$

Allowing v_i to range over different possible values at each stage could further optimize the code. With the k_i values above, we obtain a $\langle v_1, v_2, v_3, v_4 \rangle^4/(150)$ GPM code with rate

$$\frac{\log_2(v_1 \cdots v_4)}{150} = 1.518.$$

□

Table 4.2 shows the rates of low-complexity WOM codes and position modulation codes, which were compared in the original work on position modulation [76]. The first two columns show the parameters and rate of low-complexity, fixed-information WOM codes that were used for comparison in [76]. The third column shows the comparable position modulation code rate for given t. Since there has been recent ongoing work on capacity-approaching WOM codes [70], [80], the parameters and rates in columns one and two are no longer the best for some values of t, but we continue to use these classical WOM codes as components in order to maintain consistency with the original position modulation analysis. The final column of the table gives rates resulting from the GPM construction. The GPM code rates were calculated using the known WOM code as the component code, and assuming the maximum value of v_i is attained for

all i. The GPM codes are variable-rate WOM codes, whereas the standard WOM codes and position modulation codes were designed to give fixed-rate WOM codes, so direct comparison is not valid.

Example 4.3.4. In this example we use the Merkx WOM code [52] as an inner code, and choose $h = 50$, $T = 9$, and assume $k_1 = 15, k_2 = 15, k_3 = 10, k_4 = 5, k_5 = 5$. On the first write for any newly activated group, the number of potential messages is $v = 7$. On the second write, the number of messages is 6, since the same message cannot be kept in the activated group (it must subsequently contain a codeword that is not a generation-one codeword). On the third write of an activated group, the number of messages again goes up to seven, since the important distinction is between newly activated and previously activated groups, not second/third generation groups. we obtain a GPM WOM code with the following parameters:

$$
\begin{aligned}
v_1 &= \binom{50}{15} \cdot 7^{15}, \\
v_2 &= 6^{15} \cdot \binom{35}{15} \cdot 7^{15}, \\
v_3 &= 7^{15} \cdot 6^{15} \cdot \binom{20}{10} \cdot 7^{10}, \\
v_4 &= 7^{30} \cdot 6^{10} \cdot \binom{10}{5} \cdot 7^{5}, \\
v_5 &= 7^{25} \cdot 6^{5} \cdot \binom{5}{5} 7^{5}, \\
v_6 &= 7^{15} \cdot 6^{5}, \\
v_7 &= 7^{15}, \\
v_8 &= 7^{10}, \\
v_9 &= 7^{5}.
\end{aligned}
$$

The sum rate is 1.973. Again, letting k_i range over various values sometimes yields a higher rate code, but there is always a tradeoff between the amount of information that can be stored in early writes and the amount of information that can be stored during future writes. For example, a greater value of k_1 limits the possible values for k_i, $i \geq 2$. □

In summary, the GPM scheme results in a code with increases rewrites, and yields codes with a wide variety of block-lengths and corresponding rates.

4.4 Coset encoding on multi-level cells

The main results for this section are two construction methods that combine the coset encoding scheme with nonbinary WOM codes to obtain codes for q-ary flash cells. Throughout this section, $q \in \mathbb{N}$, and q need not be a power of a prime. The construction methods presented here share similarities with concatenation and generalized position modulation (Section 4.3) but use covering codes for the outer code, and nonbinary WOM codes for the inner code. We show how to apply the coset encoding scheme of [8] and rewrite on the components, and detail the two-step decoding process. We illustrate our construction methods with several examples, and discuss advantages and disadvantages of these constructions.

4.4.1 Binary coset encoding

Coset encoding, a general method for obtaining a WOM code from any error-correcting code, was introduced in [8]. Let $\mathcal{C}$ be an $[N, K, D]$ binary linear code with covering radius R. Using $\mathcal{C}$, we will encode $(N - K)$ bits on N cells. The messages are associ-

ated with syndromes of $\mathcal{C}$, so there are $2^{(N-K)}$ messages on each write. The encoding process is described below.

1. To encode $\mathbf{s_1} \in \mathbb{F}_2^{(N-K)}$, write a minimum weight vector $\mathbf{y_1} \in \mathbb{F}_2^N$ with syndrome $\mathbf{s_1}$.

2. To encode the next message $\mathbf{s_2}$, find $\mathbf{y_2}$ such that $\mathbf{y_2} + \mathbf{y_1}$ has minimum weight with syndrome $\mathbf{s_2}$ and the support of $\mathbf{y_2}$ and $\mathbf{y_1}$ are disjoint.

3. Repeat this process until the encoding of a new message is no longer possible.

Definition 4.4.1. A linear error-correcting $[n, k, d]$ code $\mathcal{C}$ is called *maximal* if $\mathcal{C}$ is not a subcode of a code of length n with the same distance. Equivalently, $\mathcal{C}$ is maximal if $R(\mathcal{C}) \leq (d-1)$.

We will use $\mathbf{1}$ to denote the all ones vector and $\mathbf{0}$ to denote the all zeros vector.

The main result relating to the coset encoding method is that when $\mathcal{C}$ is a maximal code and satisfies other maximal conditions on shortened versions, then the resulting WOM code guarantees T writes of $(N-K)$ bits, where T is an expression in terms of the minimum distance and covering radius of $\mathcal{C}$, and the number of shortened versions retaining maximality. Specifically, the authors present the following theorem.

Theorem 4.4.2 (Cohen, Godlewski, Merkx, 1986)**.** *Let $\mathcal{C}$ be an $[n, k, d]R$ maximal code. If for some i with $i \leq d^{\perp}$-1,[3] its shortened versions of lengths at least $(n-i)$ remain maximal and of minimum distance d, then at least T writings of $(n-k)$ bits are guaranteed with $T = 2 + \lfloor (i-R)/(d-1) \rfloor$.*

Due to the interest in nonbinary WOM codes, it is natural to ask how the coset encoding scheme works when applied to a nonbinary covering code. In this case, the

[3]Here, $d^{\perp}$ denotes the minimum distance of the dual code of $\mathcal{C}$.

rewriting capabilities come from the process of making disjoint subsets of the cells nonzero on each write, but once a cell has been programmed, it will never be reprogrammed. Thus, a major advantage of a multi-level memory—that cell levels may be increased multiple times—is not utilized. This drawback is also identified in [78], where nonbinary coset encoding is used to create efficient *binary* WOM codes. These authors and, separately, Wu [75] construct binary WOM codes using ideas similar to coset encoding on the second write of two-write constructions. These modifications avoid the maximality restrictions on the error-correcting code that make coset encoding difficult to apply to an arbitrary code. A different application of covering codes was used in the context of flash codes in [31]. The goal in that application was to obtain bounds on flash codes for large-alphabet messages using existing bounds on flash codes for small-alphabet messages.

In the following subsections, we present two methods of combining a covering code and a nonbinary WOM code to obtain a nonbinary rewriting code. The constructions share some of the features of the generalized position modulation scheme, except the outer covering code is encoded and decoded using the coset encoding scheme, and the inner code uses encoding and decoding rules of a nonbinary WOM code.

4.4.2 Construction I

Let $\mathcal{C}$ be an $[N, K, d]$ binary linear code with covering radius R, and suppose $\mathcal{C}$ is maximal. Suppose with coset encoding, $\mathcal{C}$ produces an $\langle M \rangle^T / N$ binary WOM code where $M = 2^{(N-K)}$. Let $\mathcal{W}$ be an $\langle m \rangle_q^t / n$ WOM code on q-ary cells. For both constructions, assume that codewords in $\mathcal{W}$ have distinguishable generations, and that the all-zeros word is not a codeword in $\mathcal{W}$.

We construct a length Nn q-ary WOM code that guarantees Tt writes as follows.

View the Nn cells of the memory state vector as N groups of n cells. During the writing process, each group will be in a state of "active", "active saturated", or "inactive". Let x_{ij} denote the number of groups activated on the $((i-1)T+j)^{th}$ write[4]. An "i-saturated vector" will be a word of length n that is not in $\mathcal{W}$, and is "less than" any word in generation $i+1$ of $\mathcal{W}$. Given a codeword $\mathbf{c} \in \mathcal{W}$ such that $\mathbf{c}$ is in generation i, the existence of an i-saturated vector $\mathbf{w}^i$ such that $\mathbf{c} < \mathbf{w}^i$ will be a requirement in the first construction detailed next.

The encoding for Construction I is as follows.

1. Given one of $M = 2^{(N-K)}$ messages, coset-encoding using $\mathcal{C}$ produces a length N binary word to be viewed as an indicator vector for which groups will be activated. One of m symbols can now be written on each active group using $\mathcal{W}$. The first write can store $2^{(N-K)}m^{x_{11}}$ messages, where x_{11} is the weight of the memory state vector corresponding to the syndrome message of the outer covering code, and thus represents the number of groups activated in the first write.

2. On writes 2 through T, first write a 1-saturated vector on each of the active groups. Use the outer code to encode one of M messages—such an encoding activates at least one more group. Write one of m messages from $\mathcal{W}$ on each new active group. On each of these writes, $2^{(N-K)}m^{x_{1j}}$ messages can be represented in total, where x_{1j} is the number of new groups activated during the j^{th} write.

3. For $i = 1, \ldots, t-1$, on the $(iT+1)^{th}$ write, first write an iT-saturated vector on any inactive or active groups remaining after the $(iT)^{th}$ write, and call all groups "inactive". As before, any of M messages may be stored by indicating

[4]This corresponds to the j^{th} write of the outer code in the i^{th} iteration, so the inner codewords at this stage will be generation i.

a new set of active groups, and each new "active" group can store one of m messages using a generation $i+1$ word from $\mathcal{W}$. Write $(iT+1)$ can represent $2^{(N-K)}m^{x_{i+1,1}}$ messages.

4. For $i = 1, \ldots, t-1$, for writes $(iT+2)$ to $(iT+T)$, continue in the same way as Step 2, where at the end of each write, each active group is saturated using an appropriate i-saturated vector, and at the end of the $(iT+T)^{th}$ write, all groups have an i-saturated vector. On write $(iT+j)$, for $j = 2, 3, \ldots, T$, a total of $2^{(N-K)}m^{x_{i+1,j}}$ messages can be represented.

5. Writing stops once the $(Tt)^{th}$ write is complete.

The decoding proceeds as follows.

1. Create an indicator vector $\mathbf{y} \in \mathbb{F}_2^N$ that has $y_i = 1$ if group i is active or active saturated, and $y_i = 0$ if group i is inactive. The inner codewords in the active groups have generation j words from $\mathcal{W}$ for some j, while the active saturated and inactive groups have j-saturated and $(j-1)$-saturated vectors, respectively.

2. Compute the syndrome of $\mathbf{y}$ to reveal the message M that was stored by the outer code.

3. For any group that is active but not saturated, decode the corresponding inner WOM codeword using $\mathcal{W}$.

Remark 4.4.3. In order to be able to encode the maximum number of messages on write $(iT+j)$, knowledge of x_{ij} should be known beforehand. Thus in most cases, one can only assume $x_{ij} \geq 1$ and encode $2^{(N-K)}m$ messages on an arbitrary write, which gives the lower guaranteed rate in the result below. However, the more general rate is also provided in case knowledge of the coset structure and other information on the outer code messages is available.

The following sum-rate is achieved from the construction.

Theorem 4.4.4. *The method described above (Construction I) produces a q-ary Tt-write WOM code with length Nn and sum-rate*

$$r \geq \frac{Tt(N-K) + (\sum x_{ij}) \log_2(m)}{Nn}$$

where the sum is over $i = 1, \ldots, t$ and $j = 1, \ldots, T$. Note that $Tt \leq \sum x_{ij} \leq Nt$, and so in the worst case, assuming just one activated group per write,

$$r = \frac{Tt(N-K) + (Tt) \log_2(m)}{Nn}.$$

4.4.3 Construction II

Let $\mathcal{C}$ be an $[N, K, d]$ binary linear code that with coset encoding produces an $\langle M \rangle^T / N$ binary WOM code where $M = 2^{(N-K)}$. Let $\mathcal{W}$ be an $\langle m \rangle_q^t / n$ WOM code on q-ary cells with distinguishable codeword generations, and assume that the all-zeros word is not a codeword in $\mathcal{W}$.

We construct a length Nn q-ary WOM code that guarantees $T + t - 1$ writes as follows. View the Nn cells of the memory state vector as N groups of n cells. During the writing process, each group will be in a state of "active", "active saturated", or "inactive". Let x_i denote the number of new groups activated on the i^{th} write, for $i = 1, \ldots, T$, and let $\mathbf{w}$ denote the all-$(q-1)$ vector, called the saturated vector. For convenience, define $x_i = 0$ for $i \leq 0$.

The encoding is as follows.

1. Same as Step 1 in Construction I. Given one of M messages, coset encoding

using $\mathcal{C}$ produces a length N indicator vector for which groups will be activated. One of m symbols can now be written on each active group using a generation one codeword of $\mathcal{W}$. The first write can store Mm^{x_1} messages.

2. On writes i, where $i = 2, \ldots, T$, use the outer code to encode one of M messages. If the current memory state vector of a group is a generation j codeword in $\mathcal{W}$ where $j < t$, one of m messages may be stored using a generation $j+1$ codeword of $\mathcal{W}$. If the current state of a group is a generation t codeword, write the saturated vector $\mathbf{w}$ on that group. For any new groups activated by the outer code, write one of m messages from $\mathcal{W}$ using a generation one codeword. In total, write i can store $Mm^{(x_{i-t+1}+\cdots+x_i)}$ possible messages.

3. Once T writes have been completed, the outer code can no longer be used. On write $T + t - i$ where $i = 1, \ldots (t-1)$, write one of m messages on each of the $x_{T-i+1} + \cdots + x_T$ active groups remaining that have current states not in generation t, and saturate any generation t groups using $\mathbf{w}$.

The decoding for Construction II is as follows.

1. Check if there are any generation one codewords of $\mathcal{W}$ in any of the groups. If so, compute the indicator vector of all active and saturated groups, and decode the outer code message using $\mathcal{C}$. If there are no generation one inner codewords, then the message does not contain any outer code message.

2. Among the active groups that are not saturated, decode the inner codewords using $\mathcal{W}$.

The following sum-rate is achieved from the construction.

Theorem 4.4.5. *The method described above (Construction II) produces a q-ary $T+t-1$-write WOM code of length Nn and sum-rate*

$$r \geq \frac{T(N-K) + t\sum_i (x_i)\log_2(m)}{Nn}.$$

In the worst case, assuming just one new activated group per outer code write yields

$$r = \frac{T(N-K) + (Tt)\log_2(m)}{Nn}.$$

An advantage of Construction II over Construction I is the relaxed conditions on the inner WOM code. While the rate and number of writes is inferior, most WOMs can be used as the inner code in Construction II.

Remark 4.4.6. In this construction, the inactive groups remaining after the T writes of the outer code are never used. Alternatively, an indicator cell may be added to the memory state vector to indicate when the first T writes are complete. This would allow additional messages to be written on those groups using $\mathcal{W}$ on writes $T+1$ through $T+t$.

4.4.4 Codes from Constructions I and II

Constructions I and II each require specific features of the inner WOM code. The following is an example of a nonbinary WOM code that can be used as the inner code in Construction I, which requires a saturated state between each generation of writes. It is a variation of the simple q-ary t-write WOM code presented in [18] and later in [80]. The idea for two writes is to partition the alphabet for each cell into two groups, $\{0, 1, \ldots, \lfloor \frac{q}{2} \rfloor\}$ and $\{\lfloor \frac{q}{2} \rfloor + 1, \ldots, (q-1)\}$. On the first write the cells only take values from the first group, while on the second write the cell values all come

from the second group, which results in an increase of all cell levels from write one to write two. The variation we provide is to reserve the word $\mathbf{0}$ for the inactive state, the word $\lfloor \frac{q}{2} \rfloor \mathbf{1}$ for the 1-saturated state, and the word $(q-1)\mathbf{1}$ for the 2-saturated state.

$$\underbrace{0, 1, \ldots, \lfloor \frac{q}{2} \rfloor - 1, \lfloor \frac{q}{2} \rfloor}_{\text{write one alphabet}}$$

$$\underbrace{\lfloor \frac{q}{2} \rfloor, \lfloor \frac{q}{2} \rfloor + 1, \ldots, (q-2), (q-1)}_{\text{write two alphabet}}$$

Denote the write one alphabet by A_1 and the write two alphabet by A_2. Let n be the number of cells in the memory state vector of the WOM code. Note that in the variation we present, on the first write any word in $(A_1)^n$ may be written except $\mathbf{0}$ and $\lfloor \frac{q}{2} \rfloor \mathbf{1}$, which are reserved for the states "inactive" and "1-saturated", respectively. During the second write any word in $(A_2)^n$ may be written except $\lfloor \frac{q}{2} \rfloor \mathbf{1}$ and the 2-saturated state, $(q-1)\mathbf{1}$. This modification allows this WOM code to be used as the inner code in Construction I. The sum-rate of this WOM code is

$$\frac{\log_2[((\lfloor \frac{q}{2} \rfloor + 1)^n - 2)(\lfloor \frac{q}{2} \rfloor^n - 2)]}{n}.$$

This construction can be generalized to t writes, though large q is necessary. The following partition of the cell alphabet will yield a variable-rate WOM code with t writes:

$$\underbrace{0, 1, \ldots, \left\lfloor \frac{q}{t} \right\rfloor}_{\text{write 1 alphabet}},$$

$$\underbrace{\left\lfloor \frac{q}{t} \right\rfloor, \ldots, 2\left\lfloor \frac{q}{t} \right\rfloor}_{\text{write 2 alphabet}}$$

$$\vdots$$

$$\underbrace{(t-1)\left\lfloor \frac{q}{t} \right\rfloor, \ldots, (q-1)}_{\text{write t alphabet}}.$$

Note that the saturated states are of the form $\left(\left\lfloor \frac{q}{t} \right\rfloor\right)\mathbf{1}, \left(2\left\lfloor \frac{q}{t} \right\rfloor\right)\mathbf{1}$, etc. Thus, the sum rate of this inner code is

$$\frac{\log_2[((\lfloor \frac{q}{t} \rfloor + 1)^n - 2)(\lfloor \frac{q}{t} \rfloor^n - 2)^{(t-2)}((q - t\lfloor \frac{q}{t} \rfloor)^n - 2)]}{n}.$$

The $EG(3,2)$ code from Section 3.4 is suitable as an inner code for Construction II since the write generations are distinguishable from the contents of the cells. Note that a slight variation must be used in order to ensure this for all message sequences. Specifically, in the encoding procedure of the $EG(3,2)$ WOM code, disregard any rule that results in a first generation word $(\mathbf{a}, \mathbf{b})$, with both components nonzero, or a second generation word with a zero component. For example, the steps that state "if $\mathbf{b} < \mathbf{w}$, then set $\mathbf{c} = (\mathbf{0}, \mathbf{w})$," and "if $\mathbf{a} < \mathbf{v}'$, set $\mathbf{c} = (\mathbf{v}', \mathbf{0})$" (i.e., those that indicate a second generation memory state that is indistinguishable from a first generation state) should both be omitted, so that all second generation words have the form $(\mathbf{a}, \mathbf{b})$ such that $\mathbf{a} \neq \mathbf{0}$ and $\mathbf{b} \neq \mathbf{0}$. These rules were originally created to allow for possible third writes and beyond, but since the second and third generation memory state vectors are not distinguishable, we restrict to using the inner WOM code over only two writes. As we discussed above, the simple scheme is just as good, and in the context of Construction II we can use either inner code with the same basic result.

Note that in both constructions the outer binary code can come from a more general class of WOM codes than those obtained from coset encoding, but possibly at the cost of decoding complexity. Some examples of outer binary WOM codes that

result from coset encoding are given in [9]:

- A Hamming code of length $2^r - 1$ produces a $\langle 2^r \rangle^{2^{r-2}+1}/(2^r - 1)$ WOM code using the coset encoding scheme.
- The length 23 Golay WOM code produces a $\langle 2^{11} \rangle^3/23$ WOM code using the coset encoding scheme.

Table 4.3 shows parameters of codes obtained using various inner WOM codes $\mathcal{W}$ of the type given in the previous section. The outer codes are various Hamming codes using the parameters detailed above [9]. Note that the rates are given as a range, using the upper and lower bounds on rate indicated in Subsection 4.4.2. Even the upper bound generally lies well below the theoretical upper limit on rate given in [16], indicating that the use of variable rate WOM codes for the inner and outer codes could lead to an improvement in rate. One gain associated with Constructions I and II is that the number of guaranteed writes increases significantly, depending on the constituent codes. Table 4.4 shows parameters of codes obtained using Construction II.

4.4.5 Error correction

In addition to obtaining many writes, Constructions I and II may be used for error correction when the constituent WOM codes have some error correction capability. Error correction in the context of a concatenation-type scheme for WOM has been explored in [28, 23], but in the former case the outer code in the concatenation was a traditional error-correcting code rather than a binary WOM code.

Particular types of errors that occur in Constructions I and II are distinguished in the following way: errors that effect the decoding of the outer codeword are called

Outer code	Inner code	Nn	q	rate r	Tt
$\langle 8\rangle_2^3/7$	$\langle 2\rangle_6^2/1$	7	6	$3.428 \le r \le 4.571$	6
$\langle 16\rangle_2^5/15$	$\langle 2\rangle_6^2/1$	15	6	$3.333 \le r \le 4.667$	10
$\langle 8\rangle_2^3/7$	$\langle 4\rangle_{10}^3/2$	14	10	$3.214 \le r \le 4.928$	9
$\langle 16\rangle_2^5/15$	$\langle 4\rangle_{10}^3/2$	30	10	$3 \le r \le 5$	15
$\langle 8\rangle_2^3/7$	$\langle 4\rangle_{11}^2/1$	7	11	$4.285 \le r \le 6.571$	6
$\langle 16\rangle_2^5/15$	$\langle 4\rangle_{11}^2/1$	15	11	$4 \le r \le 6.667$	10
$\langle 32\rangle_2^9/31$	$\langle 4\rangle_{11}^2/1$	31	11	$4.0645 \le r \le 6.903$	18
$\langle 8\rangle_2^3/7$	$\langle 16\rangle_{11}^2/4$	28	11	$2.357 \le r \le 4.643$	6
$\langle 16\rangle_2^5/15$	$\langle 16\rangle_{11}^2/4$	60	11	$2 \le r \le 4.667$	10
$\langle 8\rangle_2^3/7$	$\langle 9\rangle_{13}^3/2$	14	13	$3.966 \le r \le 6.684$	9
$\langle 16\rangle_2^5/15$	$\langle 9\rangle_{13}^3/2$	30	13	$3.585 \le r \le 6.755$	15
$\langle 8\rangle_2^3/7$	$\langle 4\rangle_{16}^5/2$	14	16	$5.357 \le r \le 8.214$	15
$\langle 16\rangle_2^5/15$	$\langle 4\rangle_{16}^5/2$	30	16	$5 \le r \le 8.333$	25
$\langle 8\rangle_2^3/7$	$\langle 9\rangle_{17}^4/2$	14	17	$5.285 \le r \le 8.913$	12
$\langle 16\rangle_2^5/15$	$\langle 9\rangle_{17}^4/2$	30	17	$4.78 \le r \le 9.006$	20
$\langle 8\rangle_2^3/7$	$\langle 4\rangle_{22}^7/2$	14	22	$7.5 \le r \le 11.5$	21
$\langle 16\rangle_2^5/15$	$\langle 4\rangle_{22}^7/2$	30	22	$7 \le r \le 11.667$	35

Table 4.3: Parameters for various choices of inner and outer codes in Construction I.

Outer code	Inner code	Nn	q	rate r	T+t-1
$\langle 8\rangle_2^3/7$	$\langle 2\rangle_6^2/1$	7	6	$2.142 \le r \le 3.285$	4
$\langle 16\rangle_2^5/15$	$\langle 2\rangle_6^2/1$	15	6	$2.0 \le r \le 3.333$	6
$\langle 8\rangle_2^3/7$	$\langle 4\rangle_{10}^3/2$	14	10	$1.928 \le r \le 3.642$	5
$\langle 16\rangle_2^5/15$	$\langle 9\rangle_{17}^4/2$	30	17	$2.779 \le r \le 7.006$	8
$\langle 8\rangle_2^3/7$	$\langle 4\rangle_{22}^7/2$	14	22	$3.642 \le r \le 7.642$	9
$\langle 16\rangle_2^5/15$	$\langle 4\rangle_{22}^7/2$	30	22	$3.0 \le r \le 7.667$	13

Table 4.4: Parameters for various choices of inner and outer codes in Construction II.

global errors; errors that impact the inner codewords are called local errors. Global errors in the outer code can be of the type (1) active group becomes inactive, or (2) inactive becomes active. Local errors in the inner code are Hamming errors.

The following proposition gives a general result about using codes with some error correction as the inner and outer codes of the constructions.

Proposition 4.4.7. *Let $\mathcal{C}$ be the outer $\langle M \rangle_2^T / N$ E-error-correcting WOM code, and $\mathcal{W}$ be the inner e-error-correcting $\langle m \rangle_q^t / n$ WOM code. Then the code $\mathcal{C} \boxtimes \mathcal{W}$ under either Construction I or II has parameters as indicated in Theorems 4.4.4 or 4.4.5, respectively, and can correct at least $(E+1)(e+1)-1$ errors.*

Proof. The constructions detailed in Subsections 4.4.2 and 4.4.3 hold the same for the error-correcting case. What remains is to check that all patterns of $(E+1)(e+1)-1$ can be corrected. Indeed, for an inner word to be read in error, at least $e+1$ errors must occur in a subblock of length n. Further, an outer word can be corrected as long as at most E of the N blocks are in error. Thus, as long as no more than $(E+1)(e+1)-1$ cells are in error, the memory contents can be decoded correctly. □

Many local error patterns will not actually result in a global error. For example, if an active group is read in error as a different active group, a global error does not occur.

Example 4.4.8. For the outer code, consider a single-error-correcting WOM code (formed from a two-error-correcting BCH code), presented in [84]. The WOM code has parameters $\langle 6 \rangle^t / 63$, where $t \geq 4$. For the inner WOM code, use an error-correcting WOM code construction in [25] that yields a $\langle 16 \rangle_{16}^5 / 21$ WOM code that can correct three Hamming errors. Construction II provides a length-1323 WOM code that can correct seven Hamming errors with $q = 16$, and that guarantees eight writes. □

We presented two constructions that give a general framework to obtain a nonbinary WOM code from an outer binary code and an inner nonbinary code. In particular, we showed how to use the classical coset encoding scheme on an outer binary code in order to get nonbinary WOM codes from Constructions I and II. The two constructions have tradeoffs in the number of writes versus the amount of information that can be stored at each write. While Construction I allows for more writes, less information can be guaranteed at each generation of the code. Conversely, Construction II allows for more information at each generation, but fewer writes overall. An advantage of Construction II is that it carries fewer restrictions on the properties of the inner nonbinary WOM code that can be used.

Chapter 5

Binary structured bit-interleaved LDPC codes[1]

Recently, the storage capacity of flash memory devices has increased dramatically, due in part to the development of multi-level cell (MLC) flash composed of cells that can store two bits, and triple-level cell (TLC) flash composed of cells that can store three bits [75]. The physical layout of the memory (as observed in the flash memory products used in the experiments in [77]) is as follows: the cells are organized into pages, the pages are organized into blocks, and each block contains 256 pages (resp., 384 pages) of cells in MLC (resp., TLC) flash.

In MLC flash, each cell can hold one of four symbols that may be viewed as binary 2-tuples. The left-most bit is called the most significant bit (MSB) and the right-most bit is the least significant bit (LSB), as in Figure 5.1. The two bits of a single cell are distributed among different pages so that pages contain only MSBs or only LSBs. Similarly, in TLC flash, each cell can hold one of eight symbols whose

[1]Material in this chapter will appear in [26], the *IEEE Journal on Selected Areas of Communications (JSAC), Special Issue on Communication Methodologies for the Next-Generation Storage Systems.*

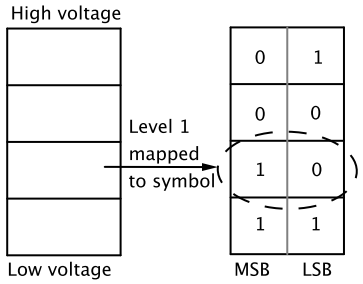

Figure 5.1: MLC flash cells and a binary mapping.

bits are distributed among MSB and LSB pages, as well as pages containing central significant bits (CSBs).

In [77, 17] the authors observe that in the TLC flash memory that they tested, a large majority of observed errors (96%) were single-bit errors, and further that the MSB pages have a lower page error rate than the CSB and LSB pages. Similar differences in MSB and LSB bit error probabilities for MLC were observed in [79]. The extent to which the bit error probability of an MSB differs from that of an LSB or CSB bit depends on the mappings of cell levels to binary representations in MLC and TLC flash that are used. An earlier analysis in [21] revealed further differences between the overall performance of SLC (single-level cell) and MLC flash products, including power, lifetime, and error rates.

Using these observations as motivation, this chapter explores how the bit assignments to MSB and LSB pages affect decoding thresholds when a binary low-density parity-check (LDPC) code is used for MLC flash. This gives insight to the optimal check node degree distributions to MSBs and LSBs for MLC flash bit-interleaved coded modulation. The degree distributions for binary LDPC codes have been shown to determine decoding performance under message-passing decoding [61, 49, 68]. In the flash memory setting, there are different bit error rates for MSBs and LSBs.

Therefore we consider the MSB-degree and LSB-degree of check nodes in the Tanner graph. The way to compare various MSB and LSB degree distributions is to compute the decoding thresholds using iterative calculations of decoding error. In general, the threshold provides a reliable indicator of the decoding performance of a particular code. Chapters 5 and 6 both use MATLAB and recursive probability-of-error calculations to determine decoding thresholds.

We consider hard decision decoding for our analysis due to both its low complexity and the fact that flash applications are aimed to provide high throughputs and access speeds. It is worth noting that several construction and decoding strategies for binary LDPC codes have been proposed [54, 44, 55] that deal with unequal error probabilities and nonuniform channels, but these are not specific to the flash memory structure.

The chapter is outlined as follows. Background and notation are presented in Section 5.1. In Section 5.2 we analyze the decoding threshold of binary regular LDPC codes for different check node to MSB and LSB degree distributions using the Gallager A and B decoding algorithms. We characterize check nodes based on the number of MSB and LSB connections (MSB and LSB degrees). Section 5.2 presents decoding thresholds in the case of graphs with two types of check node degrees. Section 5.3 presents decoding thresholds for graphs with more than two types of check node degrees. The decoding thresholds in these sections assume that different bit error probabilities are independent, which is a simplification of the physical reality, where these probabilities are in fact closely related. Therefore, Section 5.4 uses a particular signal mapping which results in related bit error probabilities, and we consider an AWGN model to obtain decoding thresholds in terms of the variance of the noise.

5.1 Motivation from MLC flash memory

In MLC flash memory, the two bits representing a symbol in a cell are stored on two different pages, one having only MSBs and one having only LSBs. One common method of storing data is to take a binary code and arbitrarily assign the bits to MSB and LSB pages, for example in an alternating fashion (Figure 5.2). This bit-interleaved coded modulation approach allows any binary code to be applicable on MLC flash. In this way, the two bits that compose a symbol are encoded independently but readable from the stored symbol. An alternative approach is to use multi-level coding in which a message is split in half and two (possibly different) codes are used on each page type (Figure 5.3). Again, this implementation uses binary codes for each page, but allows for a code with better error correction capabilities to be used on the pages with higher bit error rate.

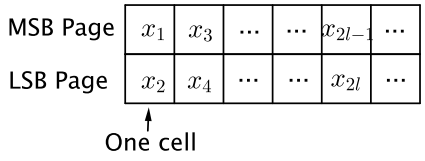

Figure 5.2: Bit-interleaved coded modulation in MLC flash cells.

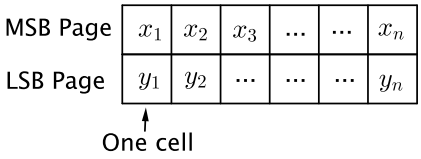

Figure 5.3: multi-level coding in MLC flash cells.

For an LDPC code used in the bit-interleaved method for MLC flash, it is natural to ask whether the number of MSB and LSB neighbors of each check node impacts the decoding performance, particularly when the voltages representing the symbols in the cells result in a greater disparity between the MSB and LSB bit error rates. In the next section we will investigate this question for binary (j, k)-regular LDPC codes in which each variable node has degree j and each check node has degree k. We say a check node has *type* $T(\alpha, \beta)$ if it has α MSB neighbors and β LSB neighbors.

Clearly, $\alpha + \beta$ is the degree of the check node.

5.2 Bit assignments for binary regular LDPC codes

In this section, we assume a binary (j, k)-regular LDPC code and examine different combinations of check nodes types to determine which has the best decoding threshold based on density evolution [19, 60, 62]. We focus on the case where the code has two types of check nodes. For $0 \leq g \leq 1$, let g be the fraction of check nodes having type $T(\alpha_1, \beta_1)$ and $(1 - g)$ be the fraction of check nodes having type $T(\alpha_2, \beta_2)$. Our convention in this section is to consider cases where $\alpha_1 \leq \alpha_2$ to avoid repetition. Let ℓ be the number of check nodes. Since half of the variable nodes are necessarily assigned to MSB pages and the other half to LSB pages, the following constraint holds:

$$\alpha_1 g\ell + \alpha_2(1-g)\ell = \frac{k\ell}{2} = \frac{\#\text{ edges}}{2}. \tag{5.2.1}$$

Consequently, $\alpha_2 = (\frac{k}{2} - \alpha_1 g)\frac{1}{1-g}$, and observe that $\beta_1 = k - \alpha_1$ and $\beta_2 = k - \alpha_2$.

For a given (j, k)-regular "cycle-free" LDPC code with two check node types as above, we will analyze the probability, as a function of the decoding iteration, that a message from a variable node to a check node is in error using the Gallager A and B algorithms [19]. Let b_1 and b_2 be the initial channel probability of error of an MSB and an LSB bit, respectively, and assume that $b_1 < b_2$.

Remark 5.2.1. An analysis of results when $b_2 < b_1$ is analogous, and can be obtained by simply reversing the roles of the MSB and LSB bits. When $b_1 = b_2$ the result is

the standard case where all bits have the same probability of error.

The Gallager A hard decision message passing algorithm requires all of the check node neighbors of a given variable node v to agree (except the neighbor c that v is sending to) in order to change the value that v sends to c in the next iteration. When calculating the probability that the message sent from a variable node to a check node in the $(t+1)^{th}$ decoding iteration is incorrect, we must consider two cases: either the variable node is an MSB, denoted v_M, or the variable node is an LSB, denoted v_L. Furthermore, denote the probability that a message sent from v_M to a neighboring check node on the $(t+1)^{th}$ iteration is in error by $p_M^{(t+1)}$, and define $p_L^{(t+1)}$ similarly. Finally, let $q_M^{(t)}$ and $q_L^{(t)}$ denote the probability that a message sent on iteration t from a check node to an MSB or LSB, respectively, is in error.

Using calculations analogous to those in [19, 48] for a (j,k)-regular graph having girth at least $2t$, we obtain the following.

Proposition 5.2.2. *If x_1 and x_2 are the number of MSB and LSB neighbors*[2]*, respectively, involved in a message update at a check node c, then the probability that the message from c is in error on iteration t is*

$$q^{(t)} = \frac{1-(1-2p_M^{(t)})^{x_1}(1-2p_L^{(t)})^{x_2}}{2}.$$

Moreover,

$$p_M^{(t+1)} = b_1\left(1-\left(1-q^{(t)}\right)^{j-1}\right)+(1-b_1)\left(q^{(t)}\right)^{j-1}, \text{ and}$$

$$p_L^{(t+1)} = b_2\left(1-\left(1-q^{(t)}\right)^{j-1}\right)+(1-b_2)\left(q^{(t)}\right)^{j-1}.$$

[2]Here, $x_1+x_2=k-1$ since the check node has degree k and the variable node receiving the message is not included in the message update.

Proof. Assume that c has α MSB neighbors and β LSB neighbors. If c is sending a message to an MSB neighbor v_M, then $x_1 = \alpha - 1$ and $x_2 = \beta$. Likewise, if the message is being sent from c to v_L, then $x_1 = \alpha$ and $x_2 = \beta - 1$. Denote by $p_{Y_i}^{(t)}$ the probability that the i^{th} neighbor, v_{Y_i}, of c sends an incorrect message on the t^{th} iteration, where $Y_i \in \{M, L\}$ for $i = 1, \ldots, k-1$. Note that c sends an incorrect message exactly when an odd number of neighboring variable nodes are incorrect. Let $g(x)$ be the generating function in which the coefficient of x^l records the probability that exactly l neighbors of c are incorrect on iteration t:

$$g(x) = \prod_{i=1}^{k-1}((1 - p_{Y_i}^{(t)}) + p_{Y_i}^{(t)}x).$$

The function $\frac{g(x)+g(-x)}{2}$ yields precisely the even powers of x. Substituting $x = 1$ into this expression gives the probability that an even number of neighbors of c send incorrect messages. Thus, the probability that c is incorrect on the t^{th} iteration is

$$\begin{aligned}
1 - \frac{g(1) + g(-1)}{2} &= 1 - \frac{\prod_{i=1}^{k-1}((1 - p_{Y_i}^{(t)}) + p_{Y_i}^{(t)}) + \prod_{i=0}^{k-1}((1 - p_{Y_i}^{(t)}) - p_{Y_i}^{(t)})}{2} \\
&= 1 - \frac{1 + \prod_{i=1}^{k-1}(1 - 2p_{Y_i}^{(t)})}{2} \\
&= \frac{1 - \prod_{i=1}^{k-1}(1 - 2p_{Y_i}^{(t)})}{2} \\
&= \frac{1 - (1 - 2p_M^{(t)})^{x_1}(1 - 2p_L^{(t)})^{x_2}}{2}.
\end{aligned}$$

The second part of the Proposition deals with the probability of an erroneous message being sent from a variable node to a check node under the Gallager A algorithm on iteration $(t+1)$. A variable node sends an incorrect message if either (1) the channel information is incorrect and at least one of the incoming check node messages is incorrect, or (2) the channel information is correct and all incoming check messages

agree and are incorrect. The expressions $p_M^{(t+1)}$ and $p_L^{(t+1)}$ above give exactly these probabilities for v_M and v_L, respectively. □

For our analysis we assume that half of the bits are MSBs and half are LSBs, and the graph has two check node types, denoted $T(\alpha_1, \beta_1)$ and $T(\alpha_2, \beta_2)$, and referred to as "Type 1" and "Type 2", respectively. The above expressions are modified accordingly, where g represents the fraction of check nodes that have Type 1. Let $q_{1,M}^{(t)}$ denote the probability that a message from a check node of Type 1 to an MSB neighbor on iteration t is in error:

$$q_{1,M}^{(t)} = \frac{(1 - (1 - 2p_M^{(t)})^{\alpha_1 - 1}(1 - 2p_L^{(t)})^{\beta_1})}{2}.$$

The error probabilities $q_{1,L}^{(t)}$, $q_{2,M}^{(t)}$, and $q_{2,L}^{(t)}$ are defined analogously. On average the probability that a message sent from a check node to an MSB neighbor on iteration t is in error is

$$q_M^{(t)} = g(q_{1,M}^{(t)}) + (1 - g)q_{2,M}^{(t)}.$$

Similarly, $q_L^{(t)} = g(q_{1,L}^{(t)}) + (1 - g)q_{2,L}^{(t)}$ is the average probability that a message sent from a check node to an LSB neighbor on iteration t is in error.

The corresponding probability of error of a message from an MSB or LSB variable node on the $(t + 1)^{th}$ iteration is

$$p_M^{(t+1)} = b_1(1 - (1 - q_M^{(t)})^{j-1}) + (1 - b_1)(q_M^{(t)})^{j-1}, \tag{5.2.2}$$

$$p_L^{(t+1)} = b_2(1 - (1 - q_L^{(t)})^{j-1}) + (1 - b_2)(q_L^{(t)})^{j-1}. \tag{5.2.3}$$

For fixed values of the MSB bit error probability b_1, we ran iterations of this

algorithm in MATLAB to determine the corresponding decoding threshold for the LSB bit error probability b_2. The *decoding threshold* is the worst-case value of b_2 such that the decoding probability of error goes to zero as the decoding iteration increases. In this case, the specific cut-off was at 100 iterations, and a probability of $p^{(100)} < 10^{-5}$ was declared a decoding success.

We considered b_1 in the range of 0.0001 to 0.1 to be fixed, and ran 100 iterations for each b_1 tested. We tested $(3, 6)$-regular, $(3, 16)$-regular $(3, 30)$-regular, and $(4, 8)$-regular LDPC codes having different check node types, and compared their decoding thresholds for b_2, including that of the standard case of a random code wherein each neighbor of each check node is equally likely to be an MSB or an LSB. The results are summarized in the following subsection. This analysis of the thresholds b_1 and b_2 for which the sequences $p_M^{(j)}$ and $p_L^{(j)}$ go to zero as $j \to \infty$ gives insight into edge label choices for the nonbinary construction in Chapter 6.

5.2.1 Results for binary (j, k)-regular codes with Gallager A

Recall that g is the fraction of check nodes of Type 1, and $(1 - g)$ is the fraction of check nodes of Type 2. We now present the results of testing the Gallager A algorithm described above for different fractions g of various check node types for $(3, 6)$-regular, $(3, 16)$-regular, $(4, 8)$-regular, and select $(3, 30)$-regular codes. Recall that once g and α_1 are fixed, the rest of α_2, β_1, and β_2 are determined. Our results indicate that for a fixed probability b_1, the best b_2 threshold occurs when $g = 1/2$ and the two check types are $T(1, k - 1)$ and $T(k - 1, 1)$ (i.e. $\alpha_1 = 1$). This suggests that codes having highly unbalanced check nodes with respect to MSBs and LSBs will perform better than the expected result of standard bit-interleaving coded modulation, which yields on average half MSB and half LSB neighbors at each check node.

Remark 5.2.3. When the check nodes are of types $T(0,k)$ and $T(k,0)$, one obtains the multi-level coding situation where MSB and LSB pages are encoded and decoded separately.

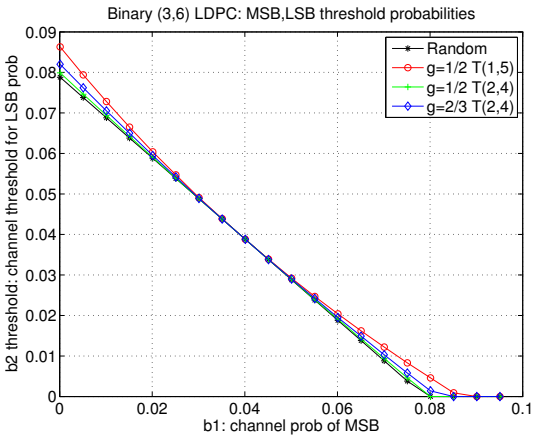

Figure 5.4: Thresholds for structured bit-interleaved $(3,6)$-regular codes and corresponding random code.

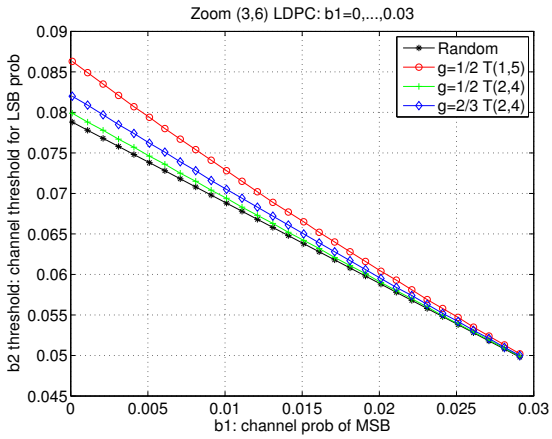

Figure 5.5: Zoom-in of Figure 5.4 to small b_1 values, specifically where $b_1 < b_2$. A higher b_2 threshold indicates a stronger code.

Figure 5.4 includes a curve for each possible combination of fraction g and Type 1 check node for a binary $(3,6)$-regular LDPC code; a close-up of the results for small values of b_1 is given in Figure 5.5. The figure legend gives the value of g and the check node type that the Type 1 check nodes have. The corresponding Type 2 for the other $(1-g)$ fraction of check nodes can be found using Equation 5.2.1. The case in which the average check node has half MSB neighbors and half LSB neighbors (denoted by "Random" in Figure 5.4) consistently has the lowest b_2 threshold, while the curve for $g = 1/2$ and Type $1 = T(1,5)$ has the highest b_2 threshold for every value of b_1. Note in this case that the corresponding Type 2 check nodes are $T(5,1)$. This implies that it is advantageous to design binary LDPC codes with more structure than using random bit assignments.

Figure 5.6 summarizes the results for binary $(3, 16)$-regular LDPC codes, where for each $\alpha_1 = 1, \ldots, 7$, only the best result is shown for clarity. For example, when $\alpha_1 = 7$, there were four values of g that yielded a legitimate value for α_2: $g = 1/2, 2/3, 3/4$, and $7/8$. We ran the simulation for each of these cases, and Figure 5.6 contains the best-performing curve which occurred when $g = 7/8$. The other values of α_1 were treated analogously. The threshold curve for the random case is also included. As is evident from Figure 5.6, the gain in b_2 threshold that is achieved by the pair $g = 1/2, T(1, 15)$ for $(3, 16)$-regular LDPC codes is the greatest, but is more subtle than the gain seen in $(3, 6)$-regular LDPC codes. We will see that this trend continues in the case of $(3, 30)$-regular LDPC codes; the higher the code rate, the smaller the gain in b_2 thresholds. However, it is notable that the extremely unbalanced check node types consistently perform among the best in all cases tested.

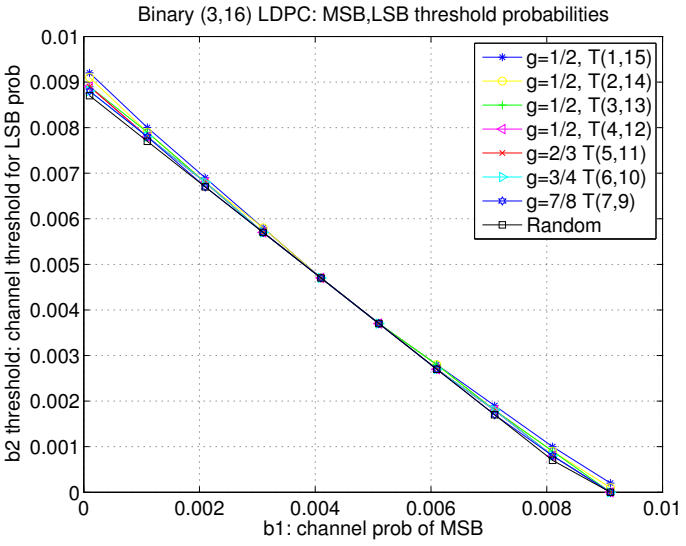

Figure 5.6: Thresholds for structured bit-interleaved $(3, 16)$-regular codes, showing the best of each $\alpha_1 = 1, \ldots, 7$.

Due to the large number of possibilities for pairs $\{g, \alpha_1\}$ when $k = 30$ (there are 60

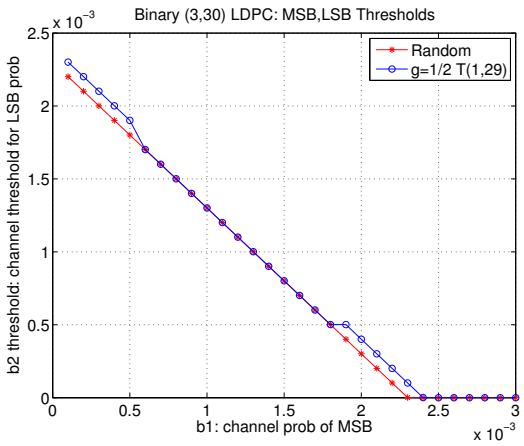

Figure 5.7: Thresholds for $(3, 30)$-regular codes, showing random vs. $g = 1/2$ and $T(1, 29)$.

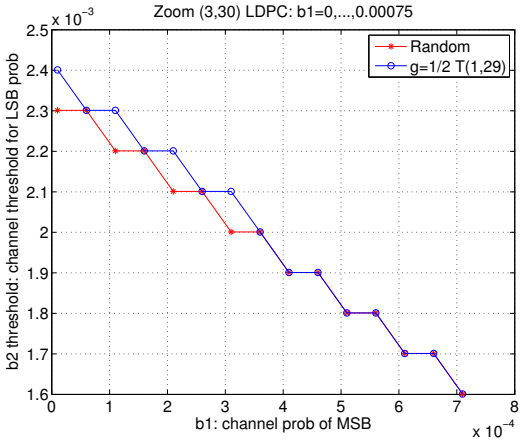

Figure 5.8: Zoom-in of Figure 5.7 to small b_1 values, where $b_1 < b_2$. Here, a finer step size in b_1 values is used than in Figure 5.7.

cases), and the fact that most of the curves lie close together, we ran the simulations for $g = 1/2, T(1, 29)$, and the random case to see whether a gain in b_2 threshold also exists in this setting. Figure 5.7 shows that for small values of b_1, the b_2 threshold in the $T(1, 29)$ case is higher than in the random case. It is important to note that the axis scales of the plot in Figure 5.7 differ from those in the other plots, because for values of $b_1 >> 2 \times 10^{-3}$, the b_2 threshold went to zero for both curves in the figure.

Similarly, we tested $(4, 8)$-regular LDPC codes for different check node types. The results are shown in Figure 5.9. Most of the cases coincided; however, the structured cases all outperformed the random case for small values of b_1. Observe that for some of the cases, the b_2 thresholds were essentially constant over certain intervals of b_1, suggesting that in these intervals, there is less dependence of b_2 on b_1 since the check node types do not have as many MSBs influencing LSBs. All of the structured bit-interleaved LDPC codes did better than the random code case where each check node had half MSB and half LSB neighbors on average.

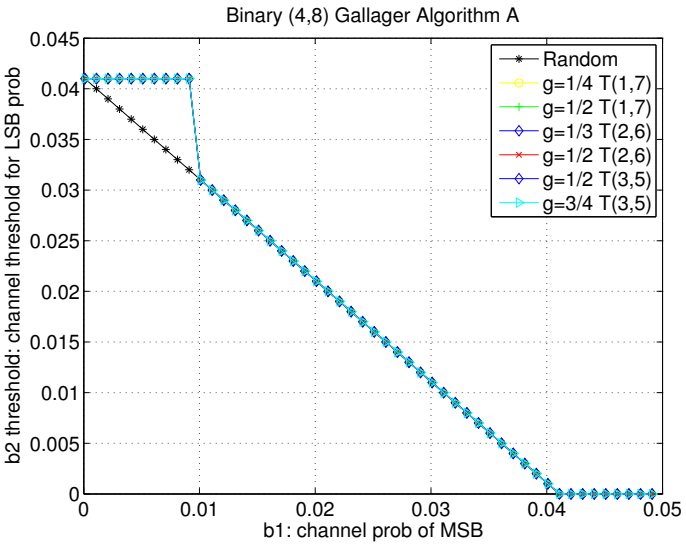

Figure 5.9: Thresholds for structured bit-interleaved $(4, 8)$-regular codes and corresponding random code.

The plots in this section used the density evolution equations presented earlier for the Gallager A algorithm applied to the two initial channel probabilities. This analysis is accurate when applied to codes with graphs of large girth. We work under this assumption since random regular bipartite graphs are known to have girth that increases logarithmically in the blocklength (see e.g., [73]).

5.2.2 Results for binary (j, k)-regular codes with Gallager B

Another hard decision decoding algorithm from [19], commonly referred to as the Gallager B algorithm, differs from the former in its update rules at the variable nodes. Rather than requiring all check messages involved in the update message to agree to change the node's estimate from that of the channel, only a majority is required. In the case of $j = 3$ and $j = 4$, the expressions for $p_M^{(t+1)}$ and $p_L^{(t+1)}$ are the same for both algorithms, and thus the results in the previous subsection are the same for Gallager B decoding. Since Gallager B typically has a better performance over Gallager A for

small check node degree, we compare the decoding thresholds for $(5, 10)$-regular and $(5, 50)$-regular LDPC codes here under both algorithms.

In Figures 5.10 and 5.11, the thresholds for structured bit-interleaved $(5, 10)$-regular LDPC codes are shown, using the Gallager A and Gallager B algorithms, respectively. As expected, the b_2 thresholds were better under Gallager B for fixed values of b_1. Under Gallager A, most of the structured codes performed comparably, but under Gallager B the best b_2 threshold was observed for the case where $g = 1/2$ of the check nodes had type $T(1, 9)$, and the other half had type $T(9, 1)$.

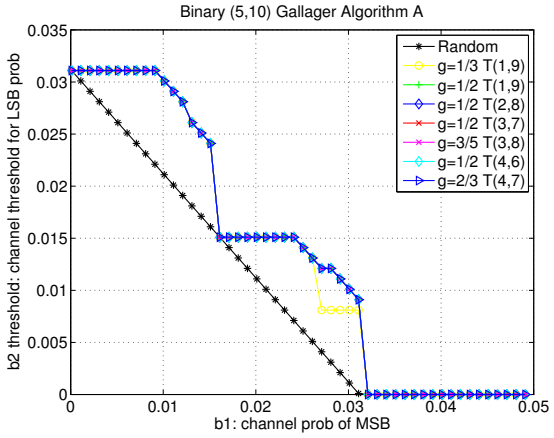

Figure 5.10: Thresholds for $(5, 10)$-regular codes, Gallager A algorithm.

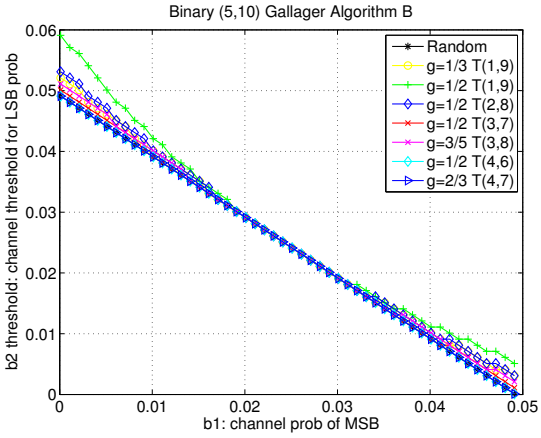

Figure 5.11: Thresholds for $(5, 10)$-regular codes, Gallager B algorithm

In Figures 5.12 and 5.13, the thresholds for $(5, 50)$-regular LDPC codes are shown using the Gallager A and Gallager B algorithms, respectively. When the check node degree is large, Gallager A is expected to perform better than Gallager B, as observed in the figures. In case A, the structured bit-interleaved codes performed noticeably better than the random bit-interleaved code.

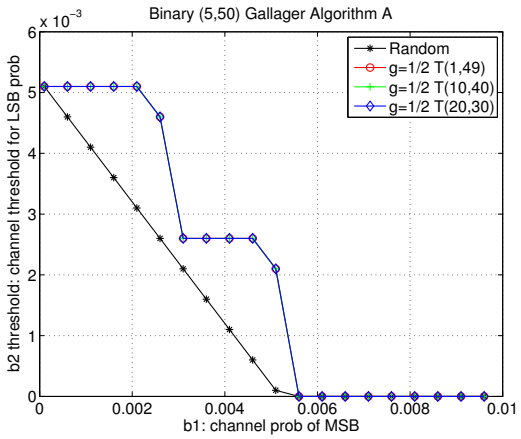

Figure 5.12: Thresholds for $(5, 50)$-regular codes, Gallager A algorithm.

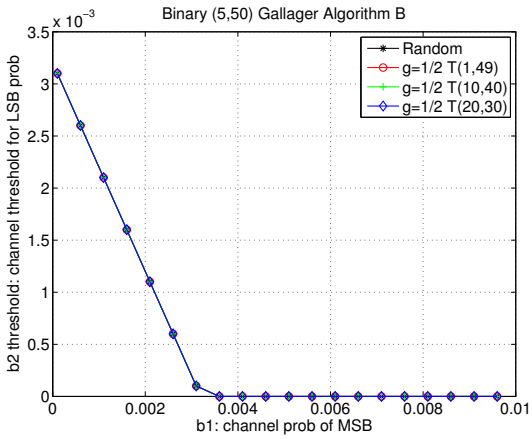

Figure 5.13: Thresholds for $(5, 50)$-regular codes, Gallager B algorithm

5.3 More than two check node types

In general, for s check types, denoted by $T(\alpha_1, \beta_1), \ldots, T(\alpha_s, \beta_s)$, let g_i be the fraction of check nodes of type i. Then $\sum_{i=1}^{s} g_i = 1$, and assuming that half of the variable nodes are MSBs and half are LSBs, the following equation holds:

$$\frac{k}{2} = g_1\alpha_1 + g_2\alpha_2 + \cdots + g_s\alpha_s.$$

For a given k and $s > 2$, there are many solutions to this equation. Although restricting the values of the g_i's yields a finite solution set, the problem of assigning bits to pages to achieve the given check node types becomes more complex as s increases.

The following general recursive equations for three check types were used to calculate the probability of decoding error after 100 iterations. The equations for q_m

and p_m are shown, and the equations for q_l and p_l are analogous.

$$q_{1m}(t) = (1 - (1 - 2p_m(t))^{(a_{1m})}(1 - 2p_l(t))^{(b_{1m})})/2$$
$$q_{2m}(t) = (1 - (1 - 2p_m(t))^{(a_{2m})}(1 - 2p_l(t))^{(b_{2m})})/2$$
$$q_{3m}(t) = (1 - (1 - 2p_m(t))^{(a_{3m})}(1 - 2p_l(t))^{(b_{3m})})/2,$$

where $a_{1m} = \alpha_1 - 1$ and $b_{1m} = \beta_1$, etc. (For $q_{1l}(t)$, $a_{1l} = \alpha_1$ and $b_{1l} = \beta_1 - 1$.)

$$q_m(t) = g_1 q_{1m}(t) + g_2 q_{2m}(t) + g_3 q_{3m}(t) \tag{5.3.1}$$

$$p_m(t+1) = b_1(1 - (1 - q_m(t))^{(j-1)}) + (1 - b_1)(q_m(t)^{(j-1)}). \tag{5.3.2}$$

In Figures 5.14, 5.15, and 5.16 we consider three check types in a $(3, 6)$-regular LDPC code. The figures show the thresholds for the possible check node types for certain fixed values of g_1, g_2, and g_3. Figure 5.17 contains thresholds for the case of four check types when the checks are evenly partitioned; again, the code is assumed to be $(3, 6)$-regular. Certain configurations of three check types outperform the best-performing configurations of two check types, as well as the four types tested in Figure 5.17.

Observation: Codes with more than half of their check nodes having a majority of MSB neighbors, i.e., $T(5, 1)$ or $T(4, 2)$, have higher thresholds than the expected BICM case.

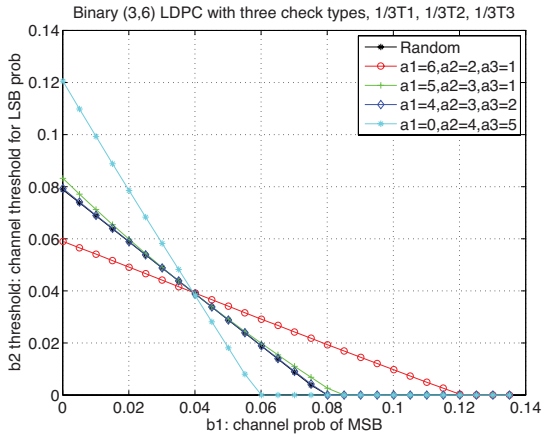

Figure 5.14: Three check types, with ratios $g_1 = g_2 = g_3 = \frac{1}{3}$.

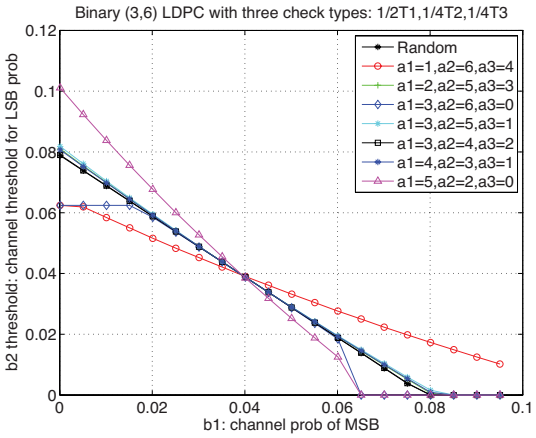

Figure 5.15: Three check types, with ratios $g_1 = \frac{1}{2}$, $g_2 = g_3 = \frac{1}{4}$.

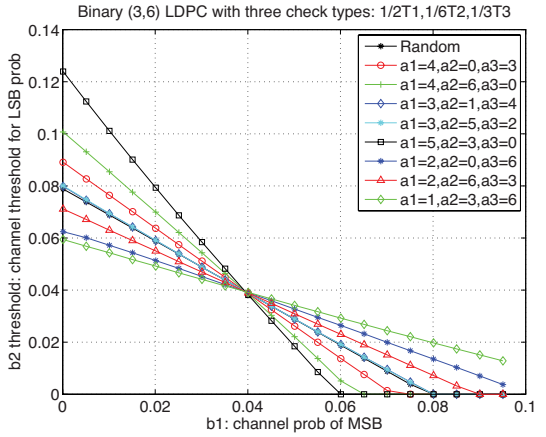

Figure 5.16: Three check types, with ratios $g_1 = \frac{1}{2}, g_2 = \frac{1}{6}, g_3 = \frac{1}{3}$.

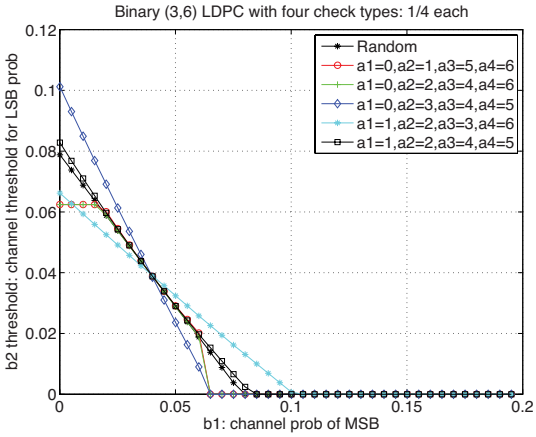

Figure 5.17: Four check types, with ratios $g_1 = g_2 = g_3 = g_4 = \frac{1}{4}$.

5.4 Results in terms of noise variance and SNR thresholds

Since the LSB and MSB probabilities of error are not independent variables in the physical setup of the memory, we extend the earlier analysis of the decoding using a particular noise model and a specific signal mapping. In this section, we assume that the noise in the memory is modeled as *additive white Gaussian noise* (AWGN),

which is a common model for noise that is attributed to random natural effects. The noise is added to the stored symbol, and it is given by a normal distribution centered at zero and with variance σ^2.

In a normal distribution, the probability that a random variable will take on a value larger than x is given by the *Q-function*:

$$Q(x) = \int_x^\infty \frac{1}{\sqrt{2\pi}} e^{-t^2/2} dt.$$

Since the noise can potentially impact both the MSB and LSB portion of the cell in the memory, we consider both parts when choosing the mapping for the symbols. The four symbols $\{11, 10, 00, 01\}$ are mapped to the 4-ary signal set $\{-3, -1, +1, +3\}$ (i.e., the cell voltage levels), and we assume that AWGN noise is added to the stored signal [58]. The cutoff between the signals is the halfway point, rounding down; i.e., if the retrieved signal is y, where $0 < y \leq 2$, then the stored signal is assumed to be $+1$, representing symbol 00.

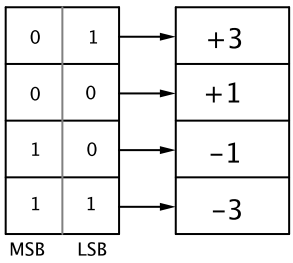

Figure 5.18: Mapping of two-bit symbols to a 4-ary signal set (cell voltage levels).

First assume that a signal $s = +3$ is stored, and the retrieved signal that is obtained while reading the cell is $y = s + n$. The probability that an MSB error occurs is $P(y \leq 0 | s = +3)$. This is equivalent to $P(n \leq -3)$. Given the iid Gaussian

pdf of the noise, we have

$$\begin{aligned} P(n \leq -3) &= \int_{-\infty}^{-3} \frac{1}{\sqrt{2\pi\sigma^2}} e^{u^2/2\sigma^2} du \\ &= \int_{-\infty}^{\frac{3}{\sigma}} \frac{1}{\sqrt{2\pi}} e^{\frac{t^2}{2}} dt \\ &= \int_{\frac{3}{\sigma}}^{\infty} \frac{1}{\sqrt{2\pi}} e^{\frac{t^2}{2}} dt \\ &= Q\left(\frac{3}{\sigma}\right), \end{aligned}$$

where the third equality is due to symmetry of the Q-function.

A difference of one cell level in the stored and retrieved symbols can be estimated by $Q(1/\sigma)$; two cell levels can be estimated by $Q(3/\sigma)$; three cell levels by $Q(5/\sigma)$. The Q-function is a decreasing function, and $Q(1/\sigma)$ is significantly larger than the other two values.

We can use a similar calculation to show that $P(y \leq 0|s = +1) = Q(\frac{1}{\sigma})$. By the symmetry of the signal set, $P(y > 0|s = -1) = Q(\frac{1}{\sigma})$, and $P(y > 0|x = -3) = Q(\frac{3}{\sigma})$. We assume that all four symbols are equally likely to be stored, and therefore we can calculate b_1 (the probability of MSB error) in terms of the noise variance

$$\begin{aligned} b_1 &= \frac{1}{4}Q\left(\frac{1}{\sigma}\right) + \frac{1}{4}Q\left(\frac{1}{\sigma}\right) + \frac{1}{4}Q\left(\frac{3}{\sigma}\right) + \frac{1}{4}Q\left(\frac{3}{\sigma}\right) \\ &= \frac{1}{2}Q\left(\frac{1}{\sigma}\right) + \frac{1}{2}Q\left(\frac{3}{\sigma}\right). \end{aligned}$$

For the LSB error calculation, we use the fact that $Q(\frac{1}{\sigma}) >> Q(\frac{5}{\sigma})$ to estimate:

$$P(\text{ LSB error } |s = +3) \approx P(y \leq 2|x = +3) = Q\left(\frac{1}{\sigma}\right).$$

If $+1$ is stored, then an LSB error has occurred if either $+3$ is retrieved or if -3

is retrieved. Therefore we obtain

$$P(\text{ LSB error } | s = +1) \approx Q\left(\frac{1}{\sigma}\right) + Q\left(\frac{3}{\sigma}\right).$$

The values for -1 and -3 are analogous. Therefore

$$P(\text{ LSB error }) \approx Q\left(\frac{1}{\sigma}\right) + Q\left(\frac{3}{\sigma}\right).$$

In summary, the MSB and LSB probabilities of error in terms of σ are given by

$$\begin{aligned} b_1 &\approx \frac{1}{2}Q\left(\frac{1}{\sigma}\right) + \frac{1}{2}Q\left(\frac{3}{\sigma}\right) \\ b_2 &\approx Q\left(\frac{1}{\sigma}\right). \end{aligned}$$

The signal-to-noise ratio is an expression in terms of the noise variance σ^2. In terms of decibals (dB), the SNR expression is

$$SNR_{dB} = 10\log_{10}\left(\frac{P_S}{P_N}\right),$$

where P_S is the power of the signal and P_N is the power of the noise. The power of AWGN is given by σ^2. With this signal set, the signal power is

$$\frac{(-1)^2 + (-3)^2 + (1)^2 + (3)^2}{4} = 5.$$

Therefore, SNR in dB is given by

$$SNR = 10\log_{10}\left(\frac{5}{\sigma^2}\right).$$

The decoding thresholds for σ and corresponding SNR thresholds (in dB) demonstrate the gain over BICM that can be achieved by using structured binary interleaving in the case of this signal set. The *threshold* for a given code is the largest noise variance (σ^2) value such that the decoding probability of error goes to zero as the iteration increases. In Table 5.1, the noise variance threshold and SNR thresholds are given for two different check node types in a $(3, 6)$-regular LDPC code. These results were obtained using MATLAB, and running the recursive system of equations (5.2.3) (5.2.2).

The rows are ordered by performance, best to worst. Table 5.2 shows the noise variance and SNR thresholds when three distinct check types are used. These results were obtained using MATLAB and the recursive system of equations (5.3.1) and (5.3.2). The gain over the expected BICM σ threshold is notably greater for the best-case of the three check types than the best case of the two check types. There are also some ratios of three check types that perform worse than the expected BICM σ threshold. The results in Table 5.1 are consistent with the observations in Figure 5.4.

Check node types	Frac. of each type	σ thres.	SNR thres.
T(1,5), T(5,1)	$1/2, 1/2$	0.6189	11.1573
T(2,4), T(5,1)	$2/3, 1/3$	0.6180	11.1699
T(2,4), T(4,2)	$1/2, 1/2$	0.6175	11.1770
Expected for BICM	Random	0.6172	11.1812

Table 5.1: Noise variance and SNR thresholds for $(3, 6)$-regular LDPC codes with two given check types.

That is, when the MSB probability of bit error from the channel is smaller than that of the LSB, the check node configuration of $T(1, 5), T(5, 1)$ with $g_1 = g_2 = \frac{1}{2}$ has the best σ and SNR threshold. Similarly, Table 5.2 shows that the same check type configurations outperform the expected BICM case as in Figures 5.14, 5.15, and

Check node types	Frac. of each type	σ thres.	SNR thres.
T(5,1), T(3,3), T(0,6)	$1/2, 1/6, 1/3$	0.6408	10.8552
T(0,6), T(4,2), T(5,1)	$1/3, 1/3, 1/3$	0.6405	10.8593
T(4,2), T(6,0), T(0,6)	$1/2, 1/6, 1/3$	0.6323	10.9712
T(5,1), T(2,4), T(0,6)	$1/2, 1/4, 1/4$	0.6287	11.0208
T(4,2), T(0,6), T(3,3)	$1/2, 1/6, 1/3$	0.6245	11.0791
T(5,1), T(3,3), T(1,5)	$1/3, 1/3, 1/3$	0.6183	11.1657
T(3,3), T(5,1), T(1,5)	$1/2, 1/4, 1/4$	0.6180	11.1699
T(2,4), T(5,1), T(3,3)	$1/2, 1/4, 1/4$	0.6178	11.1727
T(4,2), T(3,3), T(1,5)	$1/2, 1/4, 1/4$	0.6177	11.1741
T(3,3), T(1,5), T(4,2)	$1/2, 1/6, 1/3$	0.6175	11.1770
T(3,3), T(5,1), T(2,4)	$1/2, 1/6, 1/3$	0.6175	11.1770
T(4,2), T(3,3), T(2,4)	$1/3, 1/3, 1/3$	0.6174	11.1784
T(3,3), T(4,2), T(2,4)	$1/2, 1/4, 1/4$	0.6173	11.1798
Expected for BICM	Random	0.6172	11.1812
T(3,3), T(6,0), T(0,6)	$1/2, 1/4, 1/4$	0.6169	11.1854
T(2,4), T(0,6), T(6,0)	$1/2, 1/6, 1/3$	0.6117	11.2589
T(2,4), T(6,0), T(3,3)	$1/2, 1/6, 1/3$	0.6106	11.2746
T(1,5), T(6,0), T(4,2)	$1/2, 1/4, 1/4$	0.6062	11.3374
T(1,5), T(3,3), T(6,0)	$1/2, 1/6, 1/3$	0.5986	11.4470
T(6,0), T(2,4), T(1,5)	$1/3, 1/3, 1/3$	0.5984	11.4499

Table 5.2: Noise variance and SNR thresholds for $(3, 6)$-regular LDPC codes with three given check types.

5.16. Moreover, we again see that configurations with at least half of the check nodes having type $T(5, 1)$ or $T(4, 2)$ (a majority of MSB neighbors) perform better than the expected BICM case.

These threshold results indicate that given a good binary code, a structured approach to assigning the coded bits to MSBs and LSBs can yield better results than standard BICM. An open problem in this area is to develop an algorithm that takes a given Tanner graph and assigns variable nodes to MSB and LSB pages such that the result is as close as possible to the unbalanced check node types that performed well in this chapter.

Chapter 6

Nonbinary structured bit-interleaved LDPC codes[1]

Nonbinary LDPC codes have been an important area of study since the 1990s resurgence of the work of Gallager [19]. Finding efficient ways of exploiting nonbinary codes and LDPC codes in multi-level flash memories has been a goal in the last decade since flash memory became prominent [51, 85]. More recently, there has been an increased focus on nonbinary LDPC codes for various applications [11, 56, 40, 12]. This chapter gives a general approach to designing nonbinary codes when the two bits that compose a symbol over $\mathbb{F}_4$ possess different initial bit error probabilities. This is a general phenonmenon that exists in many storage and transmission settings.

A Tanner graph derived from a nonbinary parity-check matrix $\mathcal{H}$ has an edge between the j^{th} variable node and the i^{th} check node if there is a nonzero entry in position (i, j) of $\mathcal{H}$, and the edge is labeled by the matrix entry. This chapter presents a nonbinary LDPC code construction and implementation method based partly on

[1]Material in this chapter will appear in [26], the *IEEE Journal on Selected Areas of Communications (JSAC), Special Issue on Communication Methodologies for the Next-Generation Storage Systems.*

the analysis in Chapter 5. The resulting nonbinary codes are sensitive to the different error rates between the MSB and LSB pages, and we examine how the choice of edge labels impacts the bit (and overall symbol) reliability.

This chapter begins with an introduction to the binary image of a nonbinary parity-check matrix over $\mathbb{F}_{2^m}$. We define the binary expanded graph of a nonbinary LDPC code over $\mathbb{F}_{2^m}$. We then show how this representation of the code facilitates the implementation of nonbinary codes for MLC flash memory by assigning bits to MSB and LSB pages in a natural way. Previous work on nonbinary LDPC codes has also refered to the binary image of the parity-check matrix, and we use the terminology *binary image parity-check matrix* in the same way as [56]. In [56, 57], the authors chose nonzero row entries of a parity-check matrix $\mathcal{H}$ using the binary image of field elements to improve performance, assuming that the positions of the nonzero entries of $\mathcal{H}$ were already optimized. Binary image expansion techniques were also used in [36] to iteratively decode Reed-Solomon codes.

In Section 6.3, we use the results from Chapter 5 to choose nonbinary edge labels for the Tanner graph of a (j, k)-regular LDPC code. We present an implementation method for nonbinary LDPC codes on multi-level flash cells, where the decoding of the nonbinary code is done by using a binary hard-decision decoder on its binary expanded graph. This reduces complexity in addition to addressing the different bit error probabilities. Sections 6.4 and 6.5 contain decoding thresholds in terms of AWGN variance, using a particular mapping of 4-ary cell levels to bits. Since the binary expanded graph of a nonbinary code has small cycles which can degrade the decoding performance, we also developed a hard decision nonbinary decoding technique that avoids the impact of these cycles. Section 6.5 gives a description of the nonbinary hard decision decoding algorithm. We present noise variance threshold results of various edge label choices using this decoding technique.

6.1 The binary image of a code

We summarize the basics of the binary representation of a Galois field element here (see [50] for more detail). A primitive element $r \in \mathbb{F}_{2^m}$ is the root of a primitive polynomial $f(x) = x^m + c_{m-1}x^{m-1} + \cdots + c_0$, where $c_i \in \mathbb{F}_2$, for $i = 0, \ldots, m-1$. The binary matrix representation of r is the following $m \times m$ matrix (also called the "companion matrix"):

$$A = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 1 \\ c_0 & c_1 & \cdots & c_{m-2} & c_{m-1} \end{pmatrix},$$

with characteristic polynomial $f(x)$.

Recall that the nonzero elements of $\mathbb{F}_{2^m}$ are given by $\{r^i : i = 1, \ldots, 2^m - 1\}$. The matrix representation of r^i is A^i, so the matrix representations of the nonzero elements of $\mathbb{F}_{2^m}$ are $\{A^i : i = 1, \ldots, 2^m - 1\}$. If H is an $l \times n$ matrix over $\mathbb{F}_{2^m}$, the binary image parity-check matrix is the $ml \times mn$ matrix obtained by replacing entries of H with the $m \times m$ matrix representation.

Example 6.1.1. Let r be a root of the primitive polynomial $g(x) = x^2 + x + 1$. Then the binary matrix representation of elements of $\mathbb{F}_4$ is

$$\left\{ \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}, \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \right\}.$$

The field operations are standard matrix addition and matrix multiplication. □

Example 6.1.2. Consider $\mathbb{F}_8$, with r a root of the primitive polynomial $f(x) = x^3 + x + 1$. The binary matrix representation of r is

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

The field $\mathbb{F}_8$ is composed of the elements $\{0, A, A^2, \ldots, A^7\}$ (where 0 is the 3×3 all-zero matrix and A^7 is the identity matrix), under the standard matrix operations. □

In Example 6.1.1 the $GF(4)$ representation is unique, but for other $m \geq 3$, the representation depends on the choice of primitive polynomial.

Definition 6.1.3. The *binary expanded graph* of a code is the Tanner graph obtained from the binary image parity-check matrix.

Even if the original graph is regular, the binary expanded graph is usually irregular.

6.2 Implementing nonbinary codes in MLC flash

We now introduce a way to implement codes over $\mathbb{F}_4$ in MLC flash using the binary image representation. The binary expanded graph is treated as a binary LDPC code whose variable nodes represent the bits of the corresponding symbols and are assigned to different page types. Thus, the binary image of a code of length n over $\mathbb{F}_4$ gives an immediate mapping of bits to MSB pages and LSB pages. Specifically, if v_i is a variable node in the original graph over $\mathbb{F}_4$ for $i = 1, \ldots, n$, then v_{iM} and v_{iL} are the bit nodes in the binary expansion of the symbol represented by v_i, and v_{iM} is assigned to an MSB page and v_{iL} to an LSB page. To obtain a simple decoder, we will use

a binary decoder on the binary expanded graph to estimate the LSB and MSB bit values of the nonbinary symbols.

Example 6.2.1. Figure 6.1 shows the graph of a nonbinary LDPC code over $\mathbb{F}_4$ on the left, and its corresponding binary expanded graph on the right. Each variable and check node in the graph of a nonbinary LDPC code over $GF(2^m)$ splits into m copies in the binary expanded graph. For MLC flash implementation, we consider codes over $\mathbb{F}_{2^2}$ as shown in Figure 6.1 and label the copies of a variable node v_i by v_{iM} and v_{iL} to designate the bit to be assigned to an MSB or LSB page, respectively.

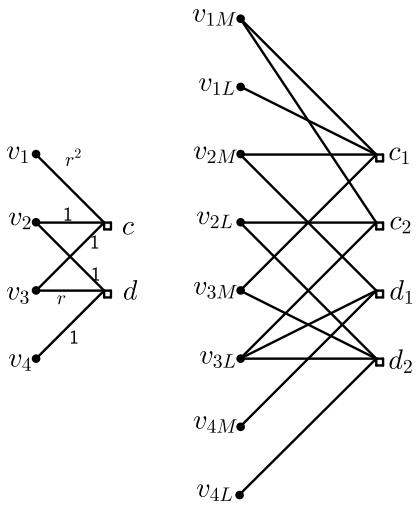

Figure 6.1: Nonbinary and binary expanded graph representations of a code over $\mathbb{F}_4$

□

In the next section, we construct nonbinary codes using underlying (j, k)-regular graphs by adding edge labels from $\mathbb{F}_4$ that give the desired types of check nodes using the analysis from Chapter 5. Note that adding nonbinary edge labels to an arbitrary (j, k)-regular graph results in a binary expanded graph with left degrees from the

set $\{j, j+1, \ldots, mj\}$ and right degrees from the set $\{k, k+1, \ldots, mk\}$. The binary expanded graph of a code over $\mathbb{F}_4$ has binary check nodes c_{i1} and c_{i2} for each $\mathbb{F}_4$ check node c_i in the nonbinary code's graph (where $i = 1, \ldots, \ell$). By choosing consistent sets of labels for the edges at each check node in the original graph, all of the check nodes labeled c_{i1} in the binary expanded graph will have the same type (as defined in Section 6.1), as will all of the check nodes labeled c_{i2}. More specifically, when choosing edge labels for a (j, k)-regular graph, we can fix a set of labels $\{r_1, \ldots, r_k\}$, where $r_i \in \mathbb{F}_4$ such that at each check node, these k labels are randomly ordered and assigned to its incident edges. Figure 6.2 shows how the labels assigned to the edges of a check node give rise to two check node types in the binary expanded graph.

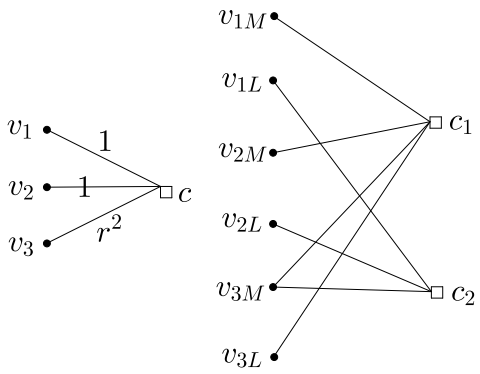

Figure 6.2: The left graph has edge labels from $\mathbb{F}_4$. The binary expanded graph on the right has check c_1 of type $T(3, 1)$ and check c_2 of type $T(1, 2)$, and is irregular since $\alpha_1 + \beta_1 \neq \alpha_2 + \beta_2$.

Remark 6.2.2. Binary images may be used for nonbinary LDPC codes over $\mathbb{F}_{2^m}$, resulting in up to m different types of check nodes in the binary expanded graph. Depending on the edge labels, some types may be the same. To design codes for TLC flash memory, the analysis in the previous sections can be extended to binary

expanded graphs with three different check node types. This can then be used to choose edge labels from $\mathbb{F}_{2^3}$ that result in the desired three check node types.

6.3 Designing codes with nonbinary edge labels

In this section, we examine how different assignments of nonbinary elements from $\mathbb{F}_4$ to the edges in an underlying regular LDPC code graph result in different binary expanded graphs. In Subsection 6.3.1, we analyze nonbinary edge label sets using binary decoding on the binary expanded graph. In Section 6.4, we then study the nonbinary decoding performance of the LDPC codes, where the binary expanded graph is no longer required.

Here, we will use insights from the thresholds in Chapter 5 to identify which edge label sets are likely to yield a binary expanded graph (and corresponding code) that performs well under binary decoding. These preferred edge labels (equivalently, assignments of nonbinary elements to nonzero positions in the parity-check matrix) result in constructions of nonbinary LDPC codes over $\mathbb{F}_4$ that have structured bit assignments to the MSB and LSB pages and may be decoded using simple binary LDPC decoders on their binary expanded graphs to estimate the MSB and LSB bits of each symbol.

We start with a parity-check matrix whose locations of nonzero entries are known (and possibly optimized), but the values have yet to be determined. In this section we illustrate our construction using the parity-check matrix of a random $(3, 6)$-regular binary LDPC code and replace the ones in the matrix with structured choices of elements from $\mathbb{F}_4$. Better codes may be obtained if a parity-check matrix with positions optimized for a nonbinary code is used. As mentioned in Section 6.2, we will assume that the field elements assigned to the edges at each check node are

Edge Label Set	Type 1	Type 2	Δ_M (MSB deg. dist.)	Δ_L (LSB deg. dist.)
$\{1, 1, A, A, A^2, A^2\}$	$T(4,4)$	$T(4,4)$	$(0, 0, \frac{8}{27}, \frac{4}{9}, \frac{2}{9}, \frac{1}{27})$	$(0, 0, \frac{8}{27}, \frac{4}{9}, \frac{2}{9}, \frac{1}{27})$
$\{1, 1, 1, A, A, A^2\}$	$T(4,3)$	$T(3,5)$	$(0, 0, \frac{125}{216}, \frac{25}{72}, \frac{5}{72}, \frac{1}{216})$	$(0, 0, \frac{8}{27}, \frac{4}{9}, \frac{2}{9}, \frac{1}{27})$
$\{1, 1, 1, A, A^2, A^2\}$	$T(5,3)$	$T(3,4)$	$(0, 0, \frac{8}{27}, \frac{4}{9}, \frac{2}{9}, \frac{1}{27})$	$(0, 0, \frac{125}{216}, \frac{25}{72}, \frac{5}{72}, \frac{1}{216})$
$\{1, 1, 1, A, A, A\}$	$T(3,3)$	$T(3,6)$	$(0, 0, 1, 0, 0, 0)$	$(0, 0, \frac{1}{8}, \frac{3}{8}, \frac{3}{8}, \frac{1}{8})$
$\{1, 1, 1, A^2, A^2, A^2\}$	$T(3,3)$	$T(6,3)$	$(0, 0, \frac{1}{8}, \frac{3}{8}, \frac{3}{8}, \frac{1}{8})$	$(0, 0, 1, 0, 0, 0)$
$\{1, 1, 1, 1, A, A^2\}$	$T(5,2)$	$T(2,5)$	$(0, 0, \frac{125}{216}, \frac{25}{72}, \frac{5}{72}, \frac{1}{216})$	$(0, 0, \frac{125}{216}, \frac{25}{72}, \frac{5}{72}, \frac{1}{216})$
$\{1, 1, 1, 1, A, A\}$	$T(4,2)$	$T(2,6)$	$(0, 0, 1, 0, 0, 0)$	$(0, 0, \frac{8}{27}, \frac{4}{9}, \frac{2}{9}, \frac{1}{27})$
$\{1, 1, 1, 1, A^2, A^2\}$	$T(6,2)$	$T(2,4)$	$(0, 0, \frac{8}{27}, \frac{4}{9}, \frac{2}{9}, \frac{1}{27})$	$(0, 0, 1, 0, 0, 0)$
$\{1, 1, 1, 1, 1, A\}$	$T(5,1)$	$T(1,6)$	$(0, 0, 1, 0, 0, 0)$	$(0, 0, \frac{125}{216}, \frac{25}{72}, \frac{5}{72}, \frac{1}{216})$
$\{1, 1, 1, 1, 1, A^2\}$	$T(6,1)$	$T(1,5)$	$(0, 0, \frac{125}{216}, \frac{25}{72}, \frac{5}{72}, \frac{1}{216})$	$(0, 0, 1, 0, 0, 0)$

Table 6.1: Edge labels for (3,6)-regular graphs and corresponding check types and degree distributions.

the same, and randomly assigned to the edges at that check node. Let $\Delta_M = (\delta_{M,1}, \delta_{M,2}, \delta_{M,3}, \delta_{M,4}, \delta_{M,5}, \delta_{M,6})$ denote the degree distribution of the MSBs, where $\delta_{M,i}$ is the fraction of MSBs having degree i, and likewise for Δ_L. Table 6.1 summarizes different sets of such edge labels, and the effect that they have on the resulting MSB degree distribution, LSB degree distribution, and check node types in the corresponding binary expanded graph.

Recall that if there are ℓ check nodes in the nonbinary code, then the check nodes of Type 1 have the form $c_{i,1}$ for $i = 1, \ldots, \ell$ in the binary expanded graph, and the check nodes of Type 2 have the form $c_{i,2}$ for $i = 1, \ldots, \ell$. Thus, in addition to providing a structured assignment of symbol bits to pages, the check nodes of each type are readily identified.

6.3.1 Performance of binary expanded graph decoding in terms of b_i thesholds

Using the degree distributions in Table 6.1, we obtain iterative equations for the expected probability of error for messages sent from variable to check nodes using the Gallager A algorithm. The probability that a check node sends a message in error to an MSB on iteration t is the expression $q_M^{(t)}$, as detailed in Chapter 5. In this setting, $g = 1/2$ and (α_1, β_1) and (α_2, β_2) are determined by the labeling of edges in the nonbinary graph (see Table 6.1, columns Type 1 and Type 2). Since the variable nodes have degree distributions given by Δ_M and Δ_L, the expressions for the MSB-to-check probability of error on the $(t+1)^{th}$ iteration is a weighted sum with $\delta_{M,i}$ coefficients. First, we derive the probability that an MSB node of degree i sends a message in error to a neighboring check: $p_{M,i}^{(t+1)} = b_1(1-(1-q_M^{(t)})^{i-1})+(1-b_1)(q_M^{(t)})^{i-1}$. Thus, the expected probability of error of an MSB-to-check message is given by

$$p_M^{(t+1)} = \sum_{i=1}^{6} \delta_{M,i} p_{M,i}^{(t+1)}. \tag{6.3.1}$$

We define $p_{L,i}^{(t+1)}$ in the analogous way to obtain

$$p_L^{(t+1)} = \sum_{i=1}^{6} \delta_{L,i} p_{L,i}^{(t+1)}. \tag{6.3.2}$$

Figure 6.3 shows the thresholds for the codes represented by the binary expanded graphs obtained from a $(3, 6)$-regular bipartite graph with the given edge label set at each check. We assume that the set of edge labels at every check node in the graph is the same (possibly permuted). For each edge label set in Table 6.1, we ran $t = 100$ iterations of the Gallager A algorithm for fixed values of the MSB error probability

b_1 to find the threshold for b_2. Due to the number of codes tested, Figure 6.3 and the following discussion focus on the range where $b_1 < b_2$.

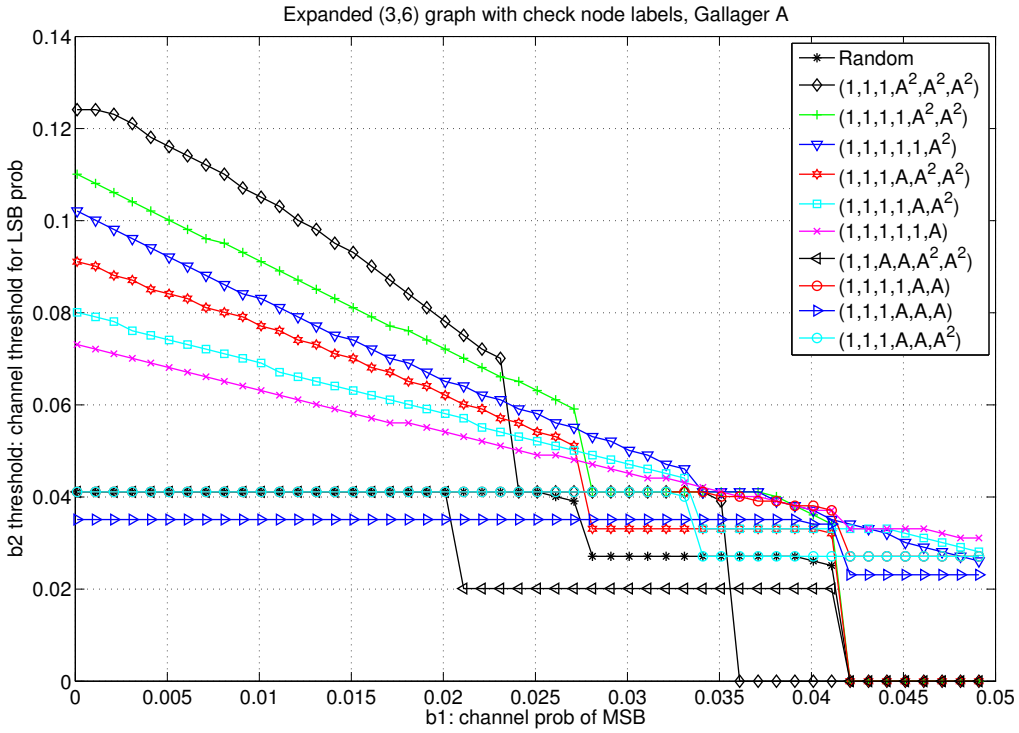

Figure 6.3: Thresholds of binary expanded graph codes obtained from $(3, 6)$-regular graphs using edge label sets from Table 6.1.

Different edge label sets result in binary codes with different degree distributions on both the variable nodes and the check nodes. The variable node degree distributions are shown in Table 6.1, whereas the check node degrees are given by the resulting check node types (i.e., half of the check nodes have degree $\alpha_1 + \beta_1$, and the rest have degree $\alpha_2 + \beta_2$). The binary expanded codes described in Table 6.1 cannot be directly compared to the (j, k)-regular codes shown in Chapter 5; however, the codes shown here may be regarded as slightly irregular, with degrees varying on fixed intervals.

Using the analysis in Chapter 5, we can determine how the best-performing check node types in the binary regular case relate to the best in configurations in the non-binary setting. Recall that codes with $g = 1/2$ of check nodes of type $T(1, k-1)$ and half of type $T(k-1, 1)$ were consistently among the best for the binary regular cases tested in Section 5.2. Thus we expect the codes with edge label sets $\{1, 1, 1, 1, 1, A^2\}$, $\{1, 1, 1, 1, 1, A\}$, and $\{1, 1, 1, 1, A^2, A^2\}$ to be the strongest since their binary images have similarly unbalanced check node types. Indeed, the codes corresponding to $\{1, 1, 1, 1, A^2, A^2\}$ and $\{1, 1, 1, 1, 1, A^2\}$ have the second and third best performance in Figure 6.3, for $0 < b_1 < 0.027$. However, $\{1, 1, 1, 1, 1, A\}$ did not perform as well, possibly due to the fact that the total number of LSB connections exceeds the number of MSB connections. Surprisingly, the best performing code was obtained using the edge labels $\{1, 1, 1, A^2, A^2, A^2\}$. While neither check node types in this case has a large difference between α_i and β_i, the total number of connections to MSB neighbors exceeds the total LSB connections more than any other case tested. An example of an edge label set yielding check types close to the random case is $\{1, 1, A, A, A^2, A^2\}$ in Table 6.1, which yields a right-regular binary expanded graph where all the check nodes have half MSB and half LSB neighbors. Due to the increase in density of edges that results from A or A^2 labels, we chose to test check label sets with at least three ones, with the exception of the "random-like" case, $\{1, 1, A, A, A^2, A^2\}$.

The curve labeled "Random" in Figure 6.3 refers to a $(3, 6)$-regular graph whose edges are randomly assigned nonzero elements of $\mathbb{F}_4$, each with equal probability. In this case, the q_M and q_L probability expression involves the degree distributions resulting from each check node edge label configuration weighted by the probability of the configuration occurring in the graph. Let ξ denote the collection of unordered check node label sets over $\mathbb{F}_4 \setminus \{0\}$ (Table 6.1 contains a partial list). Denote by $s \in \xi$ a multi-set of size six with elements from $\mathbb{F}_4 \setminus \{0\}$ resulting in binary check types

$T(\alpha_{s,1}, \beta_{s,1})$ and $T(\alpha_{s,2}, \beta_{s,2})$. Let $p(s)$ be the probability of edge set s, given that the labels are assigned randomly from $\mathbb{F}_4 \setminus \{0\}$. The density evolution expressions for the random edge assignment case are given by the following proposition.

Proposition 6.3.1. *In the binary image of a $(3, 6)$-regular graph whose edges are randomly assigned nonzero elements of $\mathbb{F}_4$, the expected probability of error for a message from a check node to an MSB or LSB variable node, respectively, on iteration t is*

$$q_M^{(t)} = \frac{1}{2} q_{1,M}^{(t)} + \frac{1}{2} q_{2,M}^{(t)}$$
$$q_L^{(t)} = \frac{1}{2} q_{1,L}^{(t)} + \frac{1}{2} q_{2,L}^{(t)}.$$

The expected probability of error of a message from a check node of type $T(\alpha_{s,1}, \beta_{s,1})$ to an MSB variable node on iteration t, denoted $q_{1,M}^{(t)}$, is

$$q_{1,M}^{(t)} = \sum_{s \in \xi} p(s) \left(\frac{1 - (1 - 2p_M^{(t)})^{\alpha_{s,1}-1} (1 - 2p_L^{(t)})^{\beta_{s,1}}}{2} \right),$$

and likewise for $q_{1,L}^{(t)}, q_{2,M}^{(t)}, q_{2,L}^{(t)}$.

Moreover, the expected probability of error for a message from an MSB (resp., LSB) variable node to a check node on iteration $(t + 1)$ is given by Equation 6.3.1 (resp., Equation 6.3.2) with $\Delta_M = \Delta_L = (0, 0, \frac{8}{27}, \frac{12}{27}, \frac{6}{27}, \frac{1}{27})$.

Proof. The expression $q_M^{(t)} = \frac{1}{2} q_{1,M}^{(t)} + \frac{1}{2} q_{2,M}^{(t)}$ requires justification. Given a random edge label from $\mathbb{F}_4 \setminus \{0\}$ on an edge $\{v, c\}$ in the nonbinary $(3, 6)$-regular graph, the binary expanded graph contains one of the corresponding subgraphs shown in Figure 6.4. Since each of the labels $1, A$, and A^2 has equal probability of being assigned to $\{v, c\}$, the probability that v_M is adjacent to c_1 in the binary expanded graph is the same

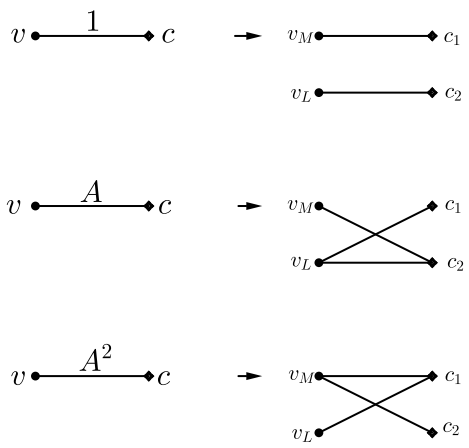

Figure 6.4: Nonzero $\mathbb{F}_4$ edge labels and the corresponding subgraphs.

as the probability that v_M is adjacent c_2 (both are 2/3, since these events are not independent). Similarly, v_L is equally likely to have c_1 as a neighbor as c_2. More precisely, we have that the probability of a random check node sending an incorrect message to an MSB node is

$$q_M^{(t)} = \frac{1}{3}q_{1,M}^{(t)} + \frac{1}{3}q_{2,M}^{(t)} + \frac{1}{3}\left(\frac{1}{2}q_{1,M}^{(t)} + \frac{1}{2}q_{2,M}^{(t)}\right) = \frac{1}{2}q_{1,M}^{(t)} + \frac{1}{2}q_{2,M}^{(t)}.$$

The rest of the proof is straightforward. □

Example 6.3.2. For example, the edge label set $\{1, 1, 1, 1, 1, A^2\}$ has a 2/243 chance of occurring in the graph, and results in two check nodes c_1 and c_2, having types $T(6, 1)$ and $T(1, 5)$, respectively. The expression $q_{1,M}^{(t)}$ described above will include the term $\frac{6}{729}(1-2p_M^{(t)})^5(1-2p_L^{(t)})$ from the configuration $\{1, 1, 1, 1, 1, A^2\}$ since $T(6, 1)$ is the resulting Type 1 check node. Similarly, the sum $q_{2,M}^{(t)}$ will include the term $\frac{6}{729}(1-2p_L^{(t)})^5$ due to the $T(1, 5)$ node c_2. □

Figure 6.5 shows the analysis for the same codes using the Gallager B algorithm,

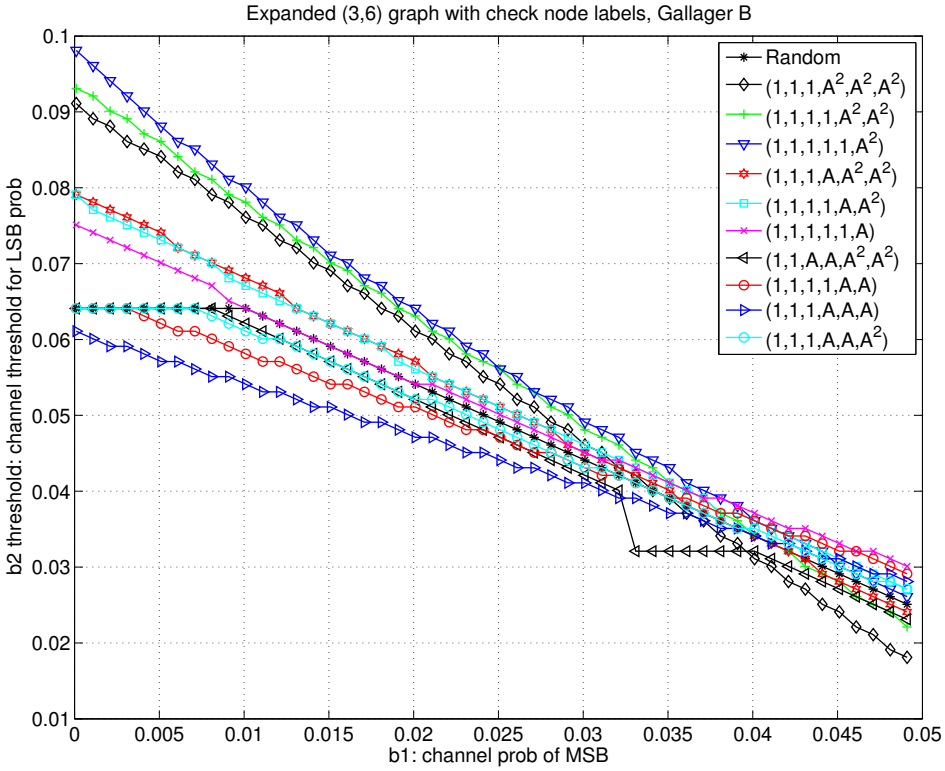

Figure 6.5: Thresholds of binary expanded graph codes obtained from $(3, 6)$-regular graphs under Gallager B decoding.

also run for $t = 100$ iterations. The probability expressions for variable nodes of degree five and six were altered to reflect Gallager B decoding. In this setting, the code with structured edge label set $\{1, 1, 1, 1, 1, A^2\}$ outperforms all other codes tested, including the random code with nonzero edge labels assigned with equal probability from $\mathbb{F}_4 \setminus \{0\}$.

Remark 6.3.3. Although we started with a random $(3, 6)$-regular graph without small cycles, the binary expanded graph most likely does contain some small local cycles, due to the introduction of subgraphs from A and A^2 (see, e.g., Figure 6.1). Since density evolution assumes the graph is cycle-free, the results in Figures 6.3

and 6.5 should be regarded as estimates. These results are still meaningful because the graph may be assumed to be globally cycle-free, as in the case of random regular LDPC codes. Edge label sets dominated by 1's will result in the least number of local cycles in the binary expanded graph. While an edge label set consisting of all ones yields a disconnected graph, the configurations we considered can be shown to be connected.

6.3.2 Nonbinary performance in terms of noise variance and SNR thresholds

In Chapter 5 we derived the initial MSB and LSB error probabilities in terms of the noise variance σ of AWGN, given the signal mapping set $\{-3, -1, 1, 3\}$:

$$b_1 \approx \frac{1}{2}Q\left(\frac{1}{\sigma}\right) + \frac{1}{2}Q\left(\frac{3}{\sigma}\right), \tag{6.3.3}$$

$$b_2 \approx Q\left(\frac{1}{\sigma}\right). \tag{6.3.4}$$

Using these values in the binary decoding analysis, we found that the σ threshold for all configurations of edge labels was the same, with the exception of the all-ones edge label set, which performed worse than the rest (see Table 6.2). The all-ones edge label set results in two disjoint graphs, one with only LSBs and one with only MSBs, which is essentially assuming that we are using the same codes in both parts of a multi-level coding scheme. Therefore, we expect the all-ones edge label set to perform worse, in general, than other edge label sets. However, the fact that all

Edge label sets	σ thres.	SNR thes. (dB)
$\{1,1,1,1,1,1\}$	0.5691	11.8859
Other	0.5774	11.7602

Table 6.2: Binary image decoding thresholds, in terms of noise variance.

other sets had the same σ and SNR thresholds indicates that the differences between the MSB and LSB connections in the binary image mappings have less impact on the binary decoding when the noise is modeled as AWGN and the initial bit error probabilities are given as in Equations 6.3.3 and 6.3.4.

Consider the line $b_1 = \frac{1}{2}b_2$ in Figures 6.3 and 6.5. This line intersects the performance curves at a location where the behavior of the curves for various edge label sets becomes erratic, indicating that this region of the (b_1, b_2) plot does not accurately capture the performance of the codes with given edge labels. Therefore, the threshold value of b_1 and b_2 in terms of noise variance is not a meaningful measure of code performance in this case.

6.4 Performance using nonbinary decoding

In this section, we will use nonbinary decoding directly on the nonbinary LDPC code graph. Nonbinary iterative decoding of LDPC codes has been shown to be efficient, although it requires more computational power than binary decoding [11]. In this section we analyze the performance of various edge label configurations under a nonbinary hard-decision message passing decoder—a generalization of Gallager's Algorithm A for codes over $\mathbb{F}_4$. We consider only $(3, 6)$-regular codes in order to assess the impact of the edge label choices that were analyzed under binary decoding in Section 6.3.

We consider the mapping given in Figure 5.18, and again assume AWGN with

variance σ^2. The probability of decoding success will be recorded in a vector of length four, where the positions correspond to the probabilities of a given message being a $0, 1, \alpha$, or α^2, respectively.

The probability vector for a message sent from a variable node to a check node on iteration t is denoted by

$$\mathbf{p}^{(t)} = (p_0^{(t)}, p_1^{(t)}, p_\alpha^{(t)}, p_{\alpha^2}^{(t)}),$$

where $p_i^{(t)}$ is the probability that a message being sent along an edge to a check node has value i.

Similarly, we define the check-to-variable probability vector $\mathbf{q}^{(t)}$:

$$\mathbf{q}^{(t)} = (q_0^{(t)}, q_1^{(t)}, q_\alpha^{(t)}, q_{\alpha^2}^{(t)}),$$

where $q_i^{(t)}$ is the probability that the check-to-variable message in iteration t has value i.

The variable node update rule is the same as in the binary case. Since we are considering only $(3, 6)$-regular codes, a variable node is processing information from the channel and two neighboring check nodes in order to send information over its remaining edge. Therefore, if both incoming check messages agree on a value in $\mathbb{F}_4$, the variable node will send that value along its third edge. If the incoming messages are distinct, then the variable node will send the message that it received from the channel.

The check node update is as follows. Five of the incoming messages from adjacent edges will be processed by the check node, and the resulting check-to-variable node message will be sent along the remaining edge. Given a particular edge label set

from Chapter 5, we will assume a uniform distribution on labels being assigned to the outgoing edge. For example, with an edge label set of $\{1, 1, 1, 1, 1, \alpha^2\}$, there is a 1/6 chance that the outgoing edge will have label α^2, and a 5/6 chance that the outgoing edge will have label 1. In the check node processing, the edge label acts as the coefficient of the incoming message when the check node is forming a linear combination of its neighboring values. Before giving the general form of the check node update, we provide a specific example.

Example 6.4.1. Consider that a check node receives the following linear combination from five of its adjacent edges:

$$1 \cdot \alpha + 1 \cdot 1 + \alpha^2 \cdot 0 + 1 \cdot \alpha + 1 \cdot 0 = 1.$$

In order for that check node to be satisfied, the message from its remaining edge would need to be the additive inverse of the sum above, which is 1 in $\mathbb{F}_4$. Therefore, the message it sends along that edge is 1 times the multiplicative inverse of the label on the outgoing edge. If the outgoing edge were labeled α, the outgoing message from the check node would be $(\alpha^{-1}) \cdot 1 = \alpha^2 \cdot 1$. □

Denote the incoming variable messages in iteration t as $\nu_1, \ldots, \nu_5 \in \mathbb{F}_4$, and the incoming edge labels as $\epsilon_1, \ldots, \epsilon_5$. Let ϵ_6 denote the outgoing edge label. Then the check node sends the following message along the outgoing edge:

$$\gamma = \epsilon_6^{-1} \sum_{i=1}^{5} \epsilon_i \nu_i.$$

The edge labels must be taken into account when calculating the updated probability vector $\mathbf{q}^{(t+1)}$. The incoming variable node probability vector $\mathbf{p}$ gets permuted when the edge label is either α or α^2. Since a variable-to-check message of 1 with

an edge label of α results in a term $1 \times \alpha$ in the check node linear combination, the $p_1(t)$ in the probability vector becomes the probability that the check node receives an α along that edge. Therefore an incoming probability vector can be permuted in the following ways:

$$(p_0^{(t)}, p_1^{(t)}, p_\alpha^{(t)}, p_{\alpha^2}^{(t)}) \xrightarrow{\alpha} (p_0^{(t)}, p_{\alpha^2}^{(t)}, p_1^{(t)}, p_\alpha^{(t)}),$$
$$(p_0^{(t)}, p_1^{(t)}, p_\alpha^{(t)}, p_{\alpha^2}^{(t)}) \xrightarrow{\alpha^2} (p_0^{(t)}, p_\alpha^{(t)}, p_{\alpha^2}^{(t)}, p_1^{(t)}).$$

In Gallager's original work [19], he derived an answer to the following question: if you have l bits, each with probability p of being a 1, what is the probability that the sum of the l bits is even? (Proposition 5.2.2 contains a modified version of this.) In the update of the check node probability vector, we will need to consider a sum of five elements of $\mathbb{F}_4$, each with probability p_i of being i, and derive the probability for each $j \in \mathbb{F}_4$ that the sum will be j. This is one step of the program in MATLAB that calculates the probability vector $\mathbf{p}^{(100)}$ that we use to assess decoding success. We use 100 decoding iterations in both Chapters 5 and 6 since the probabilities after this number of iterations provide a good indicator of decoding performance.

The following example demonstrates the process of calculating an updated $\mathbf{q}^t$ vector.

Example 6.4.2. Suppose $\beta = \lambda + \delta$, where $\lambda, \delta \in \mathbb{F}_4$. The probability vectors $L = (l_0, l_1, l_\alpha, l_{\alpha^2})$ and $D = (d_0, d_1, d_\alpha, d_{\alpha^2})$ describe the variables λ and δ, respectively; i.e., l_i is the probability that $\lambda = i$ and d_i is the probability that $\delta = i$ for $i \in \mathbb{F}_4$. Let $B = (b_0, b_1, b_\alpha, b_{\alpha^2})$ be the probability vector for β. Then in terms of L and D,

we have

$$b_0 = l_0 d_0 + l_1 d_1 + l_\alpha d_\alpha + l_{\alpha^2} d_{\alpha^2}$$

$$b_1 = l_0 d_1 + l_1 d_0 + l_\alpha d_{\alpha^2} + l_{\alpha_2} d_\alpha$$

$$b_\alpha = l_0 d_\alpha + l_1 d_{\alpha^2} + l_\alpha d_0 + l_{\alpha^2} d_1$$

$$b_{\alpha^2} = l_0 d_{\alpha^2} + l_1 d_\alpha + l_\alpha d_1 + l_{\alpha^2} d_0$$

Now suppose that $\beta' = \lambda + \delta + \tau$, where τ is described by $T = (t_0, t_1, t_\alpha, t_{\alpha^2})$. To find the probability vector for β', the equations above can be iterated, using B in place of L and T in place of D. Repeating this process three more times results in the new probability vector for a sum of five elements in $\mathbb{F}_4$. □

Since the edge label assignments are made randomly from the designated edge label set, the probability calculations will need to incorporate all possible permutations of the edge assignments. In order to visualize the process, Figure 6.6 shows three levels of the tree when the Tanner graph for the code has been unwrapped.

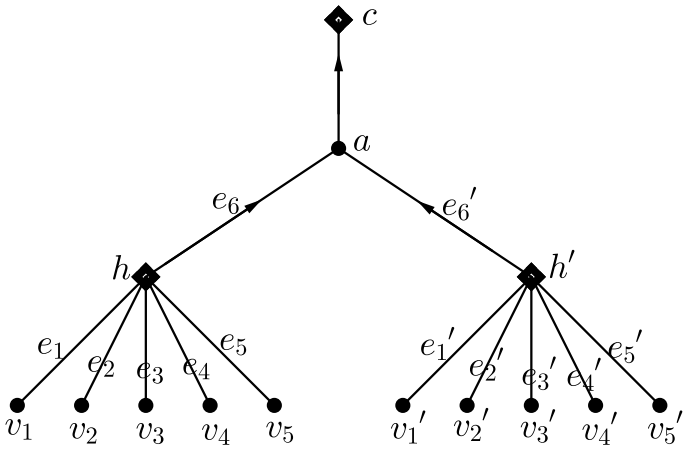

Figure 6.6: Part of the Tanner graph for a $(3, 6)$-regular code over $\mathbb{F}_4$.

The top level in Figure 6.6 is a check node labeled c. A variable node neighbor

a is sending a message along the edge $\{a, c\}$, and we will calculate the probability vector for the message being sent along that edge in iteration t.

At the outset of the calculation, the stored value for a is chosen from $\mathbb{F}_4$, where each element is equally likely. Given a particular edge label set from Section 6.3, $\{e_1, \ldots, e_6\}$ are matched to the labels in the set using a random permutation ρ of $\{1, \ldots, 6\}$. If the chosen edge label set is $\{1, 1, 1, 1, 1, \alpha^2\}$, then $e_{\rho(i)} = 1$ for $i = 1, \ldots, 5$ and $e_{\rho(6)} = \alpha^2$. A different random permutation ρ' is used to assign the labels $e'_1, \ldots, e'_6$. The stored values $v_1, \ldots, v_4$ are chosen randomly from $\mathbb{F}_4$. Since v_5 must satisfy the check node h, the value of v_5 is given by:

$$v_5 = e_5^{-1}(e_6 a + e_1 v_1 + e_2 v_2 + e_3 v_3 + e_4 v_4).$$

The same process is used for the values $v'_1, \ldots, v'_5$. The initial probability vector for vertex a depends on the assignment of the stored value from $\mathbb{F}_4$.

Proposition 6.4.3. *Given the 4-ary signal set $\{-3, -1, 1, 3\}$, and assuming AWGN with variance σ^2, the initial channel probability vectors are denoted by*

$$\mathbf{p}^{(0)}_{a=i} = (p^{(0)}_{0,i}, p^{(0)}_{1,i}, p^{(0)}_{\alpha,i}, p^{(0)}_{\alpha^2,i}),$$

where $p^{(0)}_{j,i}$ is the initial probability that j is read, given that i was stored. The proba-

bility vectors for $i = 0, 1, \alpha, \alpha^2$ *are:*

$$\mathbf{p}_{a=0}^{(0)} = \left(1 - Q\left(\frac{1}{\sigma}\right), Q\left(\frac{1}{\sigma}\right) - Q\left(\frac{3}{\sigma}\right), Q\left(\frac{3}{\sigma}\right) - Q\left(\frac{5}{\sigma}\right), Q\left(\frac{5}{\sigma}\right)\right) \text{ if } a = 0,$$
$$\mathbf{p}_{a=1}^{(0)} = \left(Q\left(\frac{1}{\sigma}\right), 1 - 2Q\left(\frac{1}{\sigma}\right), Q\left(\frac{1}{\sigma}\right) - Q\left(\frac{3}{\sigma}\right), Q\left(\frac{3}{\sigma}\right)\right) \text{ if } a = 1,$$
$$\mathbf{p}_{a=\alpha}^{(0)} = \left(Q\left(\frac{3}{\sigma}\right), Q\left(\frac{1}{\sigma}\right) - Q\left(\frac{3}{\sigma}\right), 1 - 2Q\left(\frac{1}{\sigma}\right), Q\left(\frac{1}{\sigma}\right)\right) \text{ if } a = \alpha,$$
$$\mathbf{p}_{a=\alpha^2}^{(0)} = \left(Q\left(\frac{5}{\sigma}\right), Q\left(\frac{3}{\sigma}\right) - Q\left(\frac{5}{\sigma}\right), Q\left(\frac{1}{\sigma}\right) - Q\left(\frac{3}{\sigma}\right), 1 - Q\left(\frac{1}{\sigma}\right)\right) \text{ if } a = \alpha^2.$$

Proof. We are assuming the following mapping:

$$-3 \to 0$$
$$-1 \to 1$$
$$1 \to \alpha$$
$$3 \to \alpha^2.$$

As described in Section 5.4, the probability of a difference of one cell level between the stored and retrieved symbols can be estimated by $Q(1/\sigma)$; the probability of a difference of two cell levels can be estimated by $Q(3/\sigma)$; the probability of a difference of three cell levels by $Q(5/\sigma)$. For example, given that a 1 is stored, the probability of retrieving a symbols of α is $Q(1/\sigma) - Q(3/\sigma)$. Similarly, given a stored symbol α^2, the probability of retrieving the symbol 0 is $Q(5/\sigma)$. □

The initial probability vectors for v_i and v_i' are analogous. The check node update is processed using the process described in Example 6.4.2 and the values e_i, v_i. The resulting (normalized) vectors are $\mathbf{q}^{(t)}$ and $\tilde{\mathbf{q}}^{(t)}$, from edges e_6 and e_6', respectively.

Intuitively, the probability that a variable node sends $j \in \mathbb{F}_4$ is the probability that the two incoming check messages are both j, plus the probability that the check

messages differ times the probability that the channel information at the variable node is j. We first calculate the update equations, and then normalize the probabilities to obtain $\mathbf{p}^{(t+1)}$. (The normalization guarantees that the probabilities add up to one.) The update equations at the variable node $a = i$ are:

$$\begin{aligned}
\pi_{0,i}^{(t+1)} =& q_0^{(t)}\tilde{q}_0^{(t)} + \left(1 - q_0^{(t)}\tilde{q}_0^{(t)} - q_1^{(t)}\tilde{q}_1^{(t)} - q_\alpha^{(t)}\tilde{q}_\alpha^{(t)} - q_{\alpha^2}^{(t)}\tilde{q}_{\alpha^2}^{(t)}\right) p_{0,i}^{(0)},\\
\pi_{1,i}^{(t+1)} =& q_1^{(t)}\tilde{q}_1^{(t)} + \left(1 - q_0^{(t)}\tilde{q}_0^{(t)} - q_1^{(t)}\tilde{q}_1^{(t)} - q_\alpha^{(t)}\tilde{q}_\alpha^{(t)} - q_{\alpha^2}^{(t)}\tilde{q}_{\alpha^2}^{(t)}\right) p_{1,i}^{(0)},\\
\pi_{\alpha,i}^{(t+1)} =& q_\alpha^{(t)}\tilde{q}_\alpha^{(t)} + \left(1 - q_0^{(t)}\tilde{q}_0^{(t)} - q_1^{(t)}\tilde{q}_1^{(t)} - q_\alpha^{(t)}\tilde{q}_\alpha^{(t)} - q_{\alpha^2}^{(t)}\tilde{q}_{\alpha^2}^{(t)}\right) p_{\alpha,i}^{(0)},\\
\pi_{\alpha^2,i}^{(t+1)} =& q_{\alpha^2}^{(t)}\tilde{q}_{\alpha^2}^{(t)} + \left(1 - q_0^{(t)}\tilde{q}_0^{(t)} - q_1^{(t)}\tilde{q}_1^{(t)} - q_\alpha^{(t)}\tilde{q}_\alpha^{(t)} - q_{\alpha^2}^{(t)}\tilde{q}_{\alpha^2}^{(t)}\right) p_{\alpha^2,i}^{(0)}.
\end{aligned}$$

The updated probability vector $\mathbf{p}_i^{(t+1)} = (p_{0,i}^{(t+1)}, p_{1,i}^{(t+1)}, p_{\alpha,i}^{(t+1)}, p_{\alpha^2,i}^{(t+1)})$ is given by:

$$p_{j,i}^{(t+1)} = \frac{\pi_{j,i}^{(t+1)}}{\left(\pi_{0,i}^{(t+1)} + \pi_{1,i}^{(t+1)} + \pi_{\alpha,i}^{(t+1)} + \pi_{\alpha^2,i}^{(t+1)}\right)}, \text{ for } i, j \in \mathbb{F}_4.$$

Table 6.3 shows the σ thresholds and SNR thresholds for edge label sets from Section 6.3. The results were obtained using 1000 instances of randomly chosen values for a, e_i, and e_i'. A greater number of instances would give a more accurate analysis, but the computing time for each edge label set made this difficult to obtain.

The best performing edge label set from the Gallager A decoding in Figure 6.3 remains the best in the case of nonbinary decoding: $\{1, 1, 1, \alpha^2, \alpha^2, \alpha^2\}$. However, the binary decoding analysis in Section 6.3.1 results in edge label set $\{1, 1, 1, 1, 1, \alpha^2\}$ outperforming the edge label set $\{1, 1, \alpha, \alpha, \alpha^2, \alpha^2\}$, while the nonbinary analysis has the opposite outcome. This is most likely due to the fact that the AWGN model is symmetric, which is not the case for the binary image analysis. For example, in the binary image analysis, the probability that a stored symbol of 00 would be read as

Edge label sets	σ thres.	SNR thes. (dB)
$\{1, 1, 1, \alpha^2, \alpha^2, \alpha^2\}$	0.5948	11.5023
$\{1, 1, 1, \alpha, \alpha^2, \alpha^2\}$	0.5948	11.5023
$\{1, 1, 1, 1, \alpha, \alpha^2\}$	0.5947	11.5037
$\{1, 1, \alpha, \alpha, \alpha^2, \alpha^2\}$	0.5946	11.5052
$\{1, 1, 1, 1, \alpha^2, \alpha^2\}$	0.5946	11.5052
$\{1, \alpha, \alpha, \alpha, \alpha, \alpha\}$	0.5941	11.5125
$\{1, \alpha^2, \alpha^2, \alpha^2, \alpha^2, \alpha^2\}$	0.5939	11.5154
$\{1, 1, 1, 1, 1, \alpha^2\}$	0.5937	11.5184
$\{\alpha^2, \alpha^2, \alpha^2, \alpha^2, \alpha^2, \alpha^2\}$	0.5661	11.9318

Table 6.3: Nonbinary decoding thresholds.

the symbol 10 is b_1 and the probability that it would be read as 01 is b_2. However, using the mapping given in Figure 5.18 and the AWGN model, the probabilities of the above errors are $Q(1/\sigma) - Q(3/\sigma)$ and $Q(1/\sigma)$, respectively, which does not capture the case when there are larger differences between b_1 and b_2. Finding a q-ary noise model that reflects the nature of the differing bit error probabilities remains a goal.

A distinct advantage of the nonbinary decoding method is that we are no longer concerned with cycles in the binary expanded graph[2]. Therefore it is no longer necessary to consider only edge label sets with a majority of ones. As a result we were able to test a wider variety of edge label sets, and consider the performance of label sets dominated by α and α^2 elements, although these configurations of edge labels do not perform best under the current decoding scheme.

To summarize this chapter, we first used the binary image of a graph with edge labels from $\mathbb{F}_4$ to analyze edge label sets using binary Gallager A and B decoding algorithms. We described the different check node types that result in the binary expanded graphs that we tested, and we compared these results to the expected outcome, given the good check node types in Chapter 5. We then described a nonbinary hard decision decoding algorithm and studied probability vectors, given an AWGN model.

[2]We still assume that the $(3, 6)$-regular Tanner graph is locally cycle-free.

As noted above, the strongest edge label set in both cases is $\{1, 1, 1, \alpha^2, \alpha^2, \alpha^2\}$, but there are also edge label sets whose relative performance differs among the different types of decoding.

Chapter 7

Bounds on the covering radius of graph-based codes

Families of random LDPC codes with degree sequences optimized by certain parameters tend to perform well in simulations, but these codes lack the structure needed to determine the covering radius from the Tanner graph. In [74], Wadayama considers the covering radius of the family of LDPC codes originally proposed by Gallager. In this chapter, we give bounds on the covering radius of various families of finite geometry LDPC codes. These bounds show that the covering radius of such codes grows with the size of the code, and therefore they are not promising candidates for the coset encoding WOM code construction [8, 24]. However, these results lead to new techniques in determining the covering radius, which is a classical and well-known problem for general classes of codes.

In Section 7.1 we derive a general lower bound on the covering radius of a code based on its Tanner graph. Section 7.2 provides background on constructions of finite geometry LDPC codes. In Sections 7.3 and 7.4 we derive bounds on particular families of these codes using the underlying structure of the finite geometry.

7.1 Graph-based bound on covering radius

While Tanner graphs are a common code realization tool for LDPC codes due to the efficiency of message-passing decoding, any code can be realized using a Tanner graph. Therefore, a bound on the covering radius in terms of a Tanner graph degree characterization is a useful tool for studying a variety of code classes. In the following proposition, we provide a lower bound on the covering radius of a code using a Tanner graph for the code.

Remark 7.1.1. The following proposition requires that the all ones word occurs as a syndrome of the code, which depends on the particular parity-check matrix that is used to define the code. In the special case where a parity-check matrix $\mathcal{H}$ has full rank, the all ones syndrome is guaranteed. If $\mathcal{H}$ does not have full rank, then the syndromes form an $n-k$-dimensional subspace of an M-dimensional space (where $\mathcal{H}$ has M rows), and therefore the all ones syndrome may not occur for $\mathcal{H}$.

Proposition 7.1.2. *Let $\mathcal{C}(\mathcal{H})$ be the code defined by $\mathcal{H}$, and T the Tanner graph derived from $\mathcal{H}$. Suppose M is the number of check nodes in T, and j is the maximum degree of a variable node in the Tanner graph. If the all ones word occurs as a syndrome of $\mathcal{C}(\mathcal{H})$, then the following bound holds:*

$$\frac{M}{j} \leq \mathcal{R}(\mathcal{C}),$$

Proof. Since $\mathbf{1}$ occurs as a syndrome of the code, there exists $\mathbf{x} \in \mathbb{F}_2^n$ such that $\mathbf{x}H^T = \mathbf{1}$. Let i be the minimum number of variable nodes that must be flipped so that every check node is satisfied, that is, $d(\mathbf{x}, \mathcal{C}) = i$. Since each variable node has degree at most j, we have that $M \leq ji$. Thus, $\frac{M}{j} \leq i$, and $\frac{M}{j} \leq \mathcal{R}(\mathcal{C})$. □

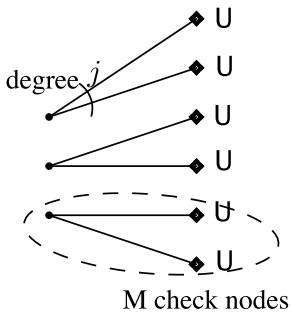

Figure 7.1: A Tanner graph model illustrating Proposition 7.1.2.

A special case of this bound is when the Tanner graph is left-j-regular, and therefore the maximum degree of a variable node in the Tanner graph is j.

A similar result was proven in [74] without the need for the all-ones syndrome, but the result only applies to the Gallager ensembles of LDPC codes. Proposition 7.1.2 applies to some families of finite geometry LDPC codes, but not all. In the following sections we derive bounds based on the incidence structure of finite geometries that are used to create families of codes and if applicable, we compare the bounds to the bound in Proposition 7.1.2.

7.2 LDPC codes from finite geometries

A linear code $\mathcal{C}$ is called *cyclic* if for every codeword $(c_1, \ldots, c_n) \in \mathcal{C}$, all cyclic shifts of the codeword are also in $\mathcal{C}$. That is,

$$(c_1, \ldots, c_n) \in \mathcal{C} \implies \{(c_2, \ldots, c_n, c_1), (c_3, \ldots, c_n, c_1, c_2), \ldots, (c_n, c_1, \ldots, c_{n-1})\} \subseteq \mathcal{C}$$

A linear code $\mathcal{C}$ is called *quasi-cyclic* if all shifts of a codeword by p positions are also in the code. When $p = 1$, the code is cyclic.

The cyclic structure allows for practical implementation of encoding and decoding

using shift registers and logic circuits. Quasi-cyclic codes can also be encoded using a shift register, and therefore they are of practical interest.

Example 7.2.1. The following example demonstrates a quasi-cyclic code with $p = 2$. Let $\mathcal{C}$ be the code generated by the matrix G.

$$G = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Observe that each row of the matrix is identical to the previous row, with a shift of two positions, and that all such shifts of the first row are present in the generator matrix. Therefore $\mathcal{C}$ has the property that for any word $\mathbf{v} \in \mathcal{C}$, all 2-position shifts of $\mathbf{v}$ are also in $\mathcal{C}$.

□

In [43], Kou, Lin, and Fossorier describe families of cyclic or quasi-cyclic LDPC codes with parity-check matrices determined by the incidence structure of finite Euclidean and projective geometries. The constructions involve defining a subgeometry without the origin point, and creating incidence matrices of points and lines for these families of subgeometries. These matrices alone can be used as parity-check matrices of LDPC codes; they can also be extended by a column splitting process that results in a code of longer length. The cyclic or quasi-cyclic structure of these codes is an advantage, however the parity-check matrices have high redundancy in the number of rows. Higher redundancy in the parity-check matrices result in increased decoding complexity, but it also has a positive effect on the decoding performance of the codes [66, 13, 41].

We recall the basic properties of finite projective and Euclidean geometries. The

m-dimensional finite projective geometry $PG(m, p^s)$ has the following parameters. There are $\rho = \frac{p^{(m+1)s}-1}{p^s-1}$ points and the number of lines is

$$(p^{ms} + \cdots + p^s + 1)(p^{(m-1)s} + \cdots + p^s + 1)/(p^s + 1).$$

Each line contains $p^s + 1$ points and each point is on $(p^{ms} - 1)/(p^s - 1)$ lines. Any two points have exactly one line in common and two lines have exactly one point in common.

The m-dimensional finite Euclidean geometry $EG(m, p^s)$ has the following parameters. There are p^{ms} points and the number of lines is

$$\frac{p^{s(m-1)}(p^{ms} - 1)}{p^s - 1}.$$

Each line contains p^s points and each point is on $\frac{p^{ms}-1}{p^s-1}$ lines. Any two points have exactly one line in common and two lines either have one point in common or are parallel.

An LDPC code can be formed from an m-dimensional finite geometry by taking the incidence matrix of μ_1-flats and μ_2-flats, where $0 \leq \mu_1 < \mu_2 \leq m$. Taking $\mu_1 = 0$ and $\mu_2 = 1$ gives the incidence matrix of points and lines in a finite geometry, which encompasses the constructions presented in [43]. However, in [43], the origin point in the Euclidean geometry is eliminated before creating the incidence matrix. In the following subsections, we include the origin point, as is the case in the more general constructions presented in [72]. *Type-I codes* use points in the geometry to correspond to columns in the parity check matrix while lines correspond to rows. *Type-II* codes have a parity check matrix that is the transpose of the Type-I parity check matrix.

We use the notation $H_{EG}^{(1)}(m, p^s)$ to denote a parity check matrix formed with the points in $EG(m, p^s)$ corresponding to columns, and lines in the geometry correspond-

ing to rows. Define $H_{EG}^{(2)}(m, p^s)$ to be a parity check matrix formed by having points in $EG(m, p^s)$ correspond to rows and lines correspond to columns. $H_{PG}^{(1)}(m, p^s)$ and $H_{PG}^{(2)}(m, p^s)$ are defined analogously. For $i = 1, 2$, the code defined by $H_{EG}^{(i)}(m, p^s)$ is denoted $\mathcal{C}_{EG}^{(i)}(m, p^s)$, and the code defined by $H_{PG}^{(i)}(m, p^s)$ is denoted $\mathcal{C}_{PG}^{(i)}(m, p^s)$.

Generalizations of finite geometry codes include codes for which a parity-check matrix is the incidence matrix of two different higher dimensional subspaces in a finite geometry [72]. For example, starting with an m-dimensional finite geometry, we can look at incidence structures of μ_2-flats and μ_1-flats, where $1 \leq \mu_1 < \mu_2 \leq m$. Creative constructions of codes using other finite incidence structures such as generalized quadrangles and latin squares have also been studied extensively [37, 41, 42].

7.3 Covering radius of Euclidean geometry LDPC codes

Our general approach to bounding the covering radius of finite Euclidean geometry LDPC codes is to consider parallel bundles of lines. The following bound for Type-I EG codes uses this strategy.

Proposition 7.3.1. *The covering radius of the Type-I Euclidean geometry LDPC code $\mathcal{C}_{EG}^{(1)}(m, 2^s)$ determined by $H_{EG}^{(1)}(m, 2^s)$ satisfies:*

$$2^{(m-1)s} \leq \mathcal{R}(\mathcal{C}_{EG}^{(1)}(m, 2^s)).$$

Proof. Recall that $H_{EG}^{(1)}(m, 2^s)$ is the incidence matrix of the Euclidean geometry $EG(m, 2^s)$, where the columns are indexed by the points and rows are indexed by the

lines of the geometry. A word $\mathbf{x} \in \mathbb{F}_2^{(2^{ms})}$ is a codeword if and only if it satisfies the following characterization. Let S be the support of $\mathbf{x}$, and let L be the set of all lines in $EG(m, 2^s)$. If S, viewed as a subset of the points of $EG(m, 2^s)$, has the property that for every line $l \in L$, the number of points from S that lie on l is even, then $\mathbf{x}$ is a codeword.

We will demonstrate a word $\mathbf{x}$ that is not a codeword, and has the additional property that $\mathbf{x}$ is a minimum weight word in its coset. Then $wt(\mathbf{x})$ will be a lower bound on the covering radius. For a given Euclidean geometry $EG(m, 2^s)$, a parallel bundle of lines is a set of parallel lines that partition the space. There are $2^{s(m-1)}$ lines in each bundle and $\frac{2^{ms}-1}{2^s-1}$ distinct parallel bundles of lines. Fix a parallel bundle of lines, and consider a word $\mathbf{x} \in \mathbb{F}_2^{(2^{ms})}$, where the support of $\mathbf{x}$ is formed by taking one point from each line in the bundle. Therefore $wt(\mathbf{x}) = 2^{s(m-1)}$. Using the observation above, we can see that in the Tanner graph of the code, there are at least $2^{s(m-1)}$ unsatisfied checks—one for each line in the parallel bundle (since each of those lines has an odd-sized intersection with the support of $\mathbf{x}$). Flipping a single variable node can alter the state of at most one of the check nodes that represents a line in the fixed parallel bundle, so in order for each of the parallel lines to become 'satisfied' in the Tanner graph, at least one variable bit per line must be flipped. The vector $\mathbf{x}$ is the minimum weight word in its coset since the indicator vector of any collection of points of size smaller than $2^{s(m-1)}$ cannot impact each of the $2^{s(m-1)}$ lines in the parallel bundle.

This gives

$$2^{(m-1)s} \leq \mathcal{R}(\mathcal{C}_{EG}^{(1)}(m, 2^s)).$$

□

We now use a similar strategy of considering a subset of points on a parallel bundle

of lines to bound the covering radius of Type-II EG codes.

Proposition 7.3.2. *The covering radius of a Type-II Euclidean geometry LDPC code* $\mathcal{C}_{EG}^{(2)}(m, 2^s)$ *determined by* $H_{EG}^{(2)}(m, 2^s)$ *satisfies:*

$$2^{(m-1)s} \leq \mathcal{R}(\mathcal{C}_{EG}^{(2)}(m, 2^s)).$$

Proof. Let L denote the number of lines in $EG(m, 2^s)$. Since $H_{EG}^{(2)}(m, 2^s) = [H_{EG}^{(1)}(m, 2^s)]^T$, codewords of $\mathcal{C}_{EG}^{(2)}(m, 2^s)$ can be characterized by lines and points in the geometry. In this case, $\mathbf{x} \in \mathbb{F}_2^{(L)}$ is a codeword if and only if for each point in the space, the number of lines in the support of $\mathbf{x}$ that pass through the point is even. A word that has ones in the positions corresponding to a bundle of parallel lines leaves every check node corresponding to a point unsatisfied. In order for the check nodes to become satisfied, the number of variable nodes that must be flipped is $2^{(m-1)s}$, since each unsatisfied check node corresponds to a point on one line in the parallel bundle. Since flipping the value of all of the variable nodes that correspond to the lines in the parallel bundle is the most efficient way to satisfy all the checks, this word is a minimum weight word in its coset. The number of unsatisfied check nodes is 2^{ms}, since every point contained in a line in the parallel bundle would be unsatisfied, and there are $2^{s(m-1)}$ parallel lines in the bundle, with 2^s points on each one. □

Remark 7.3.3. Proposition 7.1.2 can be applied to Type-II EG LDPC codes since the all-ones syndrome occurs (for example, when the support of the word corresponds to a parallel bundle of lines) and the resulting bound coincides with Proposition 7.3.2.

To refine the possible values for $\mathcal{R}(\mathcal{C}_{EG}^{(2)}(m, 2^s))$, we use the well-known *redundancy bound* [9] to provide an upper bound.

Theorem 7.3.4 (Redundancy bound)**.** *An $[n, k]$ code with covering radius $\mathcal{R}$ satisfies*

$$\mathcal{R} \leq n - k.$$

Moreover, the dimension k of various families of finite geometry LDPC codes was derived in [43, 72].

Example 7.3.5. In the case $m = 2$, we get the following bounds on the covering radius of Euclidean geometry LDPC codes:

$$2^s - 1 \leq \mathcal{R}(\mathcal{C}_{EG}^{(i)}(2, 2^s)) \leq 3^s - 1, \text{ for } i = 1, 2.$$

The lower bound comes from Propositions 7.3.1 and 7.3.2, and the upper bound from the redundancy bound. □

Another useful upper bound is the *Norse bound*, by Helleseth, Klove, and Mykkeltveit [27].

Theorem 7.3.6 (Norse bound)**.** *The covering radius of a code with zeros and ones occurring equally often in each coordinate (i.e., having dual distance at least 2) is at most $\lfloor \frac{n}{2} \rfloor$.*

None of the finite geometry LDPC codes has a zero column, and so all the codes in these families have dual distance at least two. The Norse bounds applied to the EG finite geometry codes are:

$$\mathcal{R}(\mathcal{C}_{EG}^{(1)}(m, 2^s)) \leq 2^{ms-1} - 1. \tag{7.3.1}$$

$$\mathcal{R}(\mathcal{C}_{EG}^{(2)}(m, 2^s)) \leq \frac{(2^{(m-1)s} - 1)(2^{ms} - 1)}{(2^{s+1} - 2)}. \tag{7.3.2}$$

The strategy of using the parallel structure of the Euclidean geometry has not led to upper bounds on the covering radius, but other arguments based on the incidence structure may yield more refined bounds.

7.4 Covering radius of projective geometry LDPC codes

The analysis differs when dealing with LDPC codes from projective geometries. Unlike the Euclidean geometry cases, there are no parallel bundles in finite projective spaces. To prove results about the covering radius of projective geometry code families, we will consider the distance of the vector $\mathbf{1}$ from the code. The vector $\mathbf{1}$ is not a codeword in either the Type-I or the Type-II codes, since each line in $PG(m, 2^s)$ has an odd number of points (2^s+1), and each point is contained in an odd number of lines $(\frac{2^{ms}-1}{2^s-1})$. Each check node in the respective Type-I and Type-II Tanner graphs has odd degree, and therefore the all-ones word leaves every check node unsatisfied. The distance $d(\mathbf{1}, \mathcal{C})$ provides a lower bound on the covering radius $\mathcal{R}(\mathcal{C})$, since spheres of radius $\mathcal{R}$ around codewords must cover $\mathbf{1}$.

Theorem 7.4.1 (Sphere Covering Bound)**.** *A linear $[n, k, d]$ code satisfies the following bound:*

$$\left\lfloor \frac{d-1}{2} \right\rfloor \leq \mathcal{R}(\mathcal{C}).$$

Corollary 7.4.2. *The Type-I finite projective geometry LDPC code $\mathcal{C}_{PG}^{(1)}(m, 2^s)$ has*

$$\left\lfloor \frac{2^{ms}-1}{2^{s+1}-2} \right\rfloor \leq \mathcal{R}(\mathcal{C}_{PG}^{(1)}(m, 2^s)).$$

Proof. The minimum distance of $\mathcal{C}_{PG}^{(1)}(m, 2^s)$ satisfies $\frac{(2^{ms}-1)}{(2^s-1)} + 1 \leq d_{PG}^{(1)}(m, s)$, by the

Tree bound in [73]. Therefore the result follows from the sphere covering bound. □

We now seek to improve this bound by a factor of two using the geometric structure of $PG(m, 2^s)$.

Proposition 7.4.3 (Limbupasiriporn, Storme, Vandendriessche 2012)**.** *The all-ones word* $\mathbf{1} \in \mathbb{F}_2^\rho$ *is not a codeword in* $\mathcal{C}_{PG}^{(1)}(m, 2^s)$*, and moreover, the distance from* $\mathbf{1}$ *to the code* $\mathcal{C}$ *is:*

$$d(\mathbf{1}, \mathcal{C}_{PG}^{(1)}(m, 2^s)) = \frac{2^{ms} - 1}{2^s - 1}.$$

Proof. We demonstrate a vector $\mathbf{x}$ of weight $\frac{2^{ms}-1}{2^s-1}$ such that $\mathbf{x} + \mathbf{1} \in \mathcal{C}_{PG}^{(1)}(m, 2^s)$. Fix an $m - 1$ dimensional subspace of the projective geometry $PG(m, 2^s)$, called a hyperplane, and let $\mathbf{x}$ be the indicator vector of the $\frac{2^{ms}-1}{2^s-1}$ points in the hyperplane. Every line in $PG(m, 2^s)$ is either completely in this hyperplane or intersects it in exactly one point. To see this, suppose that a line intersects the hyperplane in more than one point. There is a unique line through those two points in the hyperplane, and there is also a unique line that contains these points in $PG(m, 2^s)$. These lines must coincide, so the line is contained entirely in the hyperplane. Every check is satisfied by the word $\mathbf{x} + \mathbf{1}$, because each check node has either 2^s adjacent variable nodes with ones, or all adjacent variable nodes are zeros. Theorem 3.1 in [3] implies that a maximum weight word in the code has weight $\frac{2^{ms}-1}{2^s-1}$, so the distance is bounded below by this quantity, which then gives equality.[1] □

Corollary 7.4.4. *The covering radius of* $\mathcal{C}_{PG}^{(1)}(m, 2^s)$ *satisfies:*

$$\frac{2^{ms} - 1}{2^s - 1} \leq \mathcal{R}(\mathcal{C}_{PG}^{(1)}(m, 2^s)).$$

[1] The proof given by the authors of [47] uses a different argument.

Proof. The covering radius is bounded below by $d(\mathbf{1}, \mathcal{C}_{PG}^{(1)}(m, 2^s))$, so the result follows from Proposition 7.4.3. □

We can compare this to the graph-based result, since the all-ones syndrome occurs for the code $\mathcal{C}_{PG}^{(1)}(m, 2^s)$. Proposition 7.1.2 gives

$$\frac{2^{(m+1)s} - 1}{2^{2s} - 1} \leq \mathcal{R}(\mathcal{C}_{PG}^{(1)}(m, 2^s)).$$

The lower bound from Corollary 7.4.4 is a better bound than the one resulting from Proposition 7.1.2. Indeed,

$$\frac{2^{(m+1)s} - 1}{2^{2s} - 1} < \frac{2^{ms} - 1}{2^s - 1},$$

since the expanded version of the expression on the right contains all of the terms of the expanded expression on the left, as well as additional terms.

Example 7.4.5. In this example, we consider $m = 2$, and again use the redundancy bound and the dimension shown in [43] to determine the following range for $\mathcal{C}_{PG}^{(1)}(2, 2^s)$:

$$2^s + 1 \leq \mathcal{R}(\mathcal{C}_{PG}^{(1)}(2, 2^s)) \leq 3^s + 1.$$

In particular, the covering radius grows with the size of the underlying field of the finite projective geometry.

□

The Norse bounds also apply to the Type-I projective geometry LDPC codes, and the resulting bounds are:

$$\mathcal{R}(\mathcal{C}_{PG}^{(1)}(m, 2^s)) \leq \frac{2^{(m+1)s} - 1}{(2^{s+1} - 2)}. \tag{7.4.1}$$

$$\mathcal{R}(\mathcal{C}_{PG}^{(2)}(m,2^s)) \leq \frac{(2^{ms}+\cdots+2^s+1)(2^{(m-1)s}+\cdots+2^s+1)}{(2^{s+1}+2)}. \tag{7.4.2}$$

For Type-II PG-LDPC codes, the parity check matrix is the transpose of the corresponding Type-I parity check matrix from $PG(m,2^s)$. Recall that each point in the geometry $PG(m,2^s)$ is on

$$\frac{(2^{ms}-1)}{(2^s-1)} = 2^{(m-1)s}+2^{(m-2)s}+\cdots+2^s+1$$

lines. Every check node in the Tanner graph involves an odd number of variable nodes. The all-ones word in $\mathbb{F}_2^n$ is therefore not a codeword, and the syndrome associated with this word is the all-ones syndrome. We can mirror the process above and consider the distance from the all-ones word to the code.

Proposition 7.4.6. *The all-ones vector is not an element of $\mathcal{C}_{PG}^{(2)}(m,2^s)$, and its distance to the code is given by*

$$d(\mathbf{1},\mathcal{C}_{PG}^{(2)}(m,2^s)) = \frac{2^{ms}-1}{2^s-1}.$$

Proof. Choose a point p in $PG(m,2^s)$. Note that there are $\frac{2^{ms}-1}{2^s-1}$ lines through this point. Let $\mathbf{x}$ be the indicator vector of this set of lines. Note that $\mathbf{1}+\mathbf{x} \in \mathcal{C}$, since the check node corresponding to p is satisfied (all variable node neighbors are zero), and all other check nodes have $\frac{2^{ms}-1}{2^s-1}-1$ neighboring nodes with ones, and so are satisfied. This shows that $d(\mathbf{1},\mathcal{C}_{PG}^{(2)}(m,2^s)) \leq \frac{2^{ms}-1}{2^s-1}$. It remains to show that *at least* this many variable nodes corresponding to lines must be made zero in order to satisfy all check nodes.

Recall that the geometric interpretation of $\mathbf{1}$ is that every line has a corresponding variable node with an entry of 1. Suppose that fewer than $\frac{2^{ms}-1}{2^s-1}$ lines are changed to

zeros. Let P be the number of points contained in the union of these lines. Then

$$P \leq (2^s + 1) + \left(\frac{2^{ms} - 1}{2^s - 1} - 2 \right) (2^s),$$

since every pair of lines intersects in one point, and several lines may intersect at the same point.

Therefore, by rewriting the expression above,

$$P \leq \frac{2^{(m+1)s} - 1}{2^s - 1} + \frac{2^{s+1} - 2^s - 2^{2s}}{2^s - 1} < \frac{2^{(m+1)s} - 1}{2^s - 1}.$$

I.e., P is smaller than the number of points in the geometry. Since every point in the geometry represents a check node, and there is at least one point not contained on a line that was flipped, the distance from $\mathbf{1}$ to the code is at least $\frac{2^{ms}-1}{2^s-1}$. That is,

$$\frac{2^{ms} - 1}{2^s - 1} \leq d(\mathbf{1}, \mathcal{C}_{PG}^{(2)}(m, 2^s)).$$

□

Corollary 7.4.7. *The covering radius of* $\mathcal{C}_{PG}^{(2)}(m, 2^s)$ *satisfies:*

$$\frac{2^{ms} - 1}{2^s - 1} \leq \mathcal{R}(\mathcal{C}_{PG}^{(2)}(m, 2^s)).$$

Proof. Since the all-ones word has distance $\frac{2^{ms}-1}{2^s-1}$ from the code, the covering radius is at least as large as this distance. □

The results here suggest a general strategy for bounding the covering radius of LDPC codes derived from finite geometries—in the case of Euclidean geometries, consider parallel bundles of μ-flats, and in the case of finite projective geometries, consider the distance from the all ones word to the code. These strategies could

be applied to many of the families of finite geometry LDPC codes that have been proposed to refine existing covering radius bounds.

Chapter 8

Conclusions

This thesis has explored mathematical approaches to coding for flash memory storage. We conclude with extensions and open questions.

In Chapter 3, we constructed families of rewriting codes using the incidence structure of finite Euclidean geometries. One question that arises from these constructions is: can we rebuild a finite geometry (or more general incidence structure) using a WOM code encoding map? Can incidence structures be derived using recently-constructed WOM codes? We have investigated constructions of WOM codes using more general discrete structures, but the incidence relations of finite geometries seem to lend themselves best to deriving natural encoding and decoding maps. Perhaps starting with an efficient WOM code and creating an incidence structure can shed light on the precise incidence relations that are needed to achieve such a construction.

Extensions for Chapter 4 include classifying when a WOM code meets the lower bound given in Section 4.1. In Section 4.4, it would be interesting to calculate the optimum parameters when using error-correcting WOM codes as the inner and outer codes, and incorporating variable-rate WOM codes as the component codes.

Chapters 5 and 6 comprise an approach to the design and implementation of

binary and nonbinary LDPC codes for flash memory. One interesting question is how to take a structured binary LDPC code (such as one described in Section 7.1) and efficiently delineate the precise bit assignments in order to achieve unbalanced check node types. An alternate approach is to construct a (j, k)-regular LDPC code with the bit assignments built into the construction. These codes can then be compared to existing schemes for error-correction in flash memory (e.g., in [17]).

In Chapter 6, the binary decoding thresholds and the nonbinary decoding thresholds indicated different results for certain edge label assignments. Is there a different noise model for which these results coincide?

The results in Chapter 7 suggest that bounds on the covering radius of a wide variety of families of finite geometry LDPC codes can be derived using the geometric incidence relations. It would be interesting to extend the strategies in Chapter 7 to bound the parameters of such structured code families.

Bibliography

[1] Rudolf Ahlswede and Zhen Zhang. Coding for write-efficient memory. *Information and computation*, 83(1):80–97, 1989.

[2] Lynn Margaret Batten. *Combinatorics of finite geometries*. Cambridge University Press, 1997.

[3] RC Bose and RC Burton. A characterization of flat spaces in a finite geometry and the uniqueness of the hamming and the macdonald codes. *Journal of Combinatorial Theory*, 1(1):96–104, 1966.

[4] A. Robert Calderbank. The art of signaling: Fifty years of coding theory. *Information Theory, IEEE Transactions on*, 44(6):2561–2595, 1998.

[5] Yuval Cassuto, Moshe Schwartz, Vasken Bohossian, and Jehoshua Bruck. Codes for multi-level flash memories: Correcting asymmetric limited-magnitude errors. In *Information Theory, 2007. ISIT 2007. IEEE International Symposium on*, pages 1176–1180. IEEE, 2007.

[6] Flavio Chierichetti, Hilary Finucane, Zhenming Liu, and Michael Mitzenmacher. Designing floating codes for expected performance. *Information Theory, IEEE Transactions on*, 56(3):968–978, 2010.

[7] Gérard Cohen. On the capacity of write-unidirectional memories. *Bull. Instit. Mathemat. Academia Sinica*, 16(4):285–293, December 1988.

[8] Gérard Cohen, Philippe Godlewski, and Frans Merkx. Linear binary code for write-once memories. *Information Theory, IEEE Transactions on*, 32(5):697–700, 1986.

[9] Gérard Cohen, Iiro Honkala, Simon Litsyn, and Antoine Lobstein. *Covering codes*, volume 54. Elsevier, 1997.

[10] Matthew C Davey and David JC MacKay. Low density parity check codes over gf (q). In *Information Theory Workshop, 1998*, pages 70–71. IEEE, 1998.

[11] Matthew C Davey and David JC MacKay. Low density parity check codes over gf (q). In *Information Theory Workshop, 1998*, pages 70–71. IEEE, 1998.

[12] Dariush Divsalar and Lara Dolecek. Graph cover ensembles of non-binary protograph ldpc codes. In *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, pages 2526–2530. IEEE, 2012.

[13] Jon Feldman, Martin J Wainwright, and David R Karger. Using linear programming to decode binary linear codes. *Information Theory, IEEE Transactions on*, 51(3):954–972, 2005.

[14] Amos Fiat and Adi Shamir. Generalized'write-once'memories. *Information Theory, IEEE Transactions on*, 30(3):470–480, 1984.

[15] Hilary Finucane, Zhenming Liu, and Michael Mitzenmacher. Designing floating codes for expected performance. In *Communication, Control, and Computing, 2008 46th Annual Allerton Conference on*, pages 1389–1396. IEEE, 2008.

[16] Fang-Wei Fu and AJ Han Vinck. On the capacity of generalized write-once memory with state transitions described by an arbitrary directed acyclic graph. *Information Theory, IEEE Transactions on*, 45(1):308–313, 1999.

[17] Ryan Gabrys, Eitan Yaakobi, and Lara Dolecek. Graded bit-error-correcting codes with applications to flash memories. *IEEE Transactions on Information Theory*, 2013.

[18] Ryan Gabrys, Eitan Yaakobi, Lara Dolecek, Paul H Siegel, Alexander Vardy, and Jack K Wolf. Non-binary wom-codes for multilevel flash memories. In *Information Theory Workshop (ITW), 2011 IEEE*, pages 40–44. IEEE, 2011.

[19] Robert G Gallager. Low-density parity-check codes. *Information Theory, IRE Transactions on*, 8(1):21–28, 1962.

[20] Robert G Gallager. *Low-Density Parity-Check Codes*. MIT Press, Cambridge, MA, 1963.

[21] Laura M Grupp, Adrian M Caulfield, Joel Coburn, Steven Swanson, Eitan Yaakobi, Paul H Siegel, and Jack K Wolf. Characterizing flash memory: anomalies, observations, and applications. In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pages 24–33. IEEE, 2009.

[22] Richard W Hamming. Error detecting and error correcting codes. *Bell System technical journal*, 29(2):147–160, 1950.

[23] Kathryn Haymaker and Christine A Kelley. Coding strategies for reliable storage in multilevel flash memories. In *Proceedings of the Int'l Castle Meeting on Coding Theory and Applications*. 2011.

[24] Kathryn Haymaker and Christine A Kelley. Covering codes for multilevel flash memories. In *Proceedings of the Asilomar Conference on Signals, Systems, and Computing*, November 2012.

[25] Kathryn Haymaker and Christine A Kelley. Geometric wom codes and coding strategies for multilevel flash memories. *Designs, Codes and Cryptography: Special issue on coding theory and applications*, May 2012.

[26] Kathryn Haymaker and Christine A Kelley. Structured bit-interleaved ldpc codes for mlc flash memory. *IEEE Journal on Selected Areas of Communications (JSAC), Special Issue on Communication Methodologies for the Next-Generation Storage Systems*, May 2014.

[27] Tor Helleseth, Torleiv Kløve, and Johannes Mykkeltveit. On the covering radius of binary codes (corresp.). *Information Theory, IEEE Transactions on*, 24(5):627–628, 1978.

[28] Qin Huang, Shu Lin, and Khaled AS Abdel-Ghaffar. Error-correcting codes for flash coding. volume 57, pages 6097–6108. IEEE, 2011.

[29] Anxiao Jiang. On the generalization of error-correcting wom codes. In *Information Theory, 2007. ISIT 2007. IEEE International Symposium on*, pages 1391–1395. IEEE, 2007.

[30] Anxiao Jiang, Vasken Bohossian, and Jehoshua Bruck. Floating codes for joint information storage in write asymmetric memories. In *Information Theory, 2007. ISIT 2007. IEEE International Symposium on*, pages 1166–1170. IEEE, 2007.

[31] Anxiao Jiang, Vasken Bohossian, and Jehoshua Bruck. Rewriting codes for joint information storage in flash memories. *Information Theory, IEEE Transactions on*, 56(10):5300–5313, 2010.

[32] Anxiao Jiang and Jehoshua Bruck. Joint coding for flash memory storage. In *Information Theory, 2008. ISIT 2008. IEEE International Symposium on*, pages 1741–1745. IEEE, 2008.

[33] Anxiao Jiang and Jehoshua Bruck. Information representation and coding for flash memories. pages 920–925, 2009.

[34] Anxiao Jiang, Michael Langberg, Moshe Schwartz, and Jehoshua Bruck. Universal rewriting in constrained memories. In *Information Theory, 2009. ISIT 2009. IEEE International Symposium on*, pages 1219–1223. IEEE, 2009.

[35] Anxiao Jiang, Hao Li, and Yue Wang. Error scrubbing codes for flash memories. In *Information Theory, 2009. CWIT 2009. 11th Canadian Workshop on*, pages 32–35. IEEE, 2009.

[36] Jing Jiang and Krishna R Narayanan. Iterative soft-input soft-output decoding of reed-solomon codes by adapting the parity-check matrix. *Information Theory, IEEE Transactions on*, 52(8):3746–3756, 2006.

[37] Sarah J Johnson and Steven R Weller. Codes for iterative decoding from partial geometries. *Communications, IEEE Transactions on*, 52(2):236–243, 2004.

[38] Scott Kayser, Eitan Yaakobi, Paul H Siegel, Alexander Vardy, and Jack K Wolf. Multiple-write wom-codes. In *Communication, Control, and Computing (Allerton), 2010 48th Annual Allerton Conference on*, pages 1062–1068. IEEE, 2010.

[39] Christine A Kelley and Deepak Sridhara. Pseudocodewords of tanner graphs. *Information Theory, IEEE Transactions on*, 53(11):4013–4038, 2007.

[40] Christine A Kelley, Deepak Sridhara, and Joachim Rosenthal. Pseudocodeword weights for non-binary ldpc codes. In *Information Theory, 2006 IEEE International Symposium on*, pages 1379–1383. IEEE, 2006.

[41] Christine A Kelley, Deepak Sridhara, and Joachim Rosenthal. Tree-based construction of ldpc codes having good pseudocodeword weights. *Information Theory, IEEE Transactions on*, 53(4):1460–1478, 2007.

[42] Jon-Lark Kim, Uri N Peled, Irina Perepelitsa, Vera Pless, and Shmuel Friedland. Explicit construction of families of ldpc codes with no 4-cycles. *Information Theory, IEEE Transactions on*, 50(10):2378–2388, 2004.

[43] Yu Kou, Shu Lin, and Marc PC Fossorier. Low-density parity-check codes based on finite geometries: a rediscovery and new results. *Information Theory, IEEE Transactions on*, 47(7):2711–2736, 2001.

[44] Vidya Kumar and Olgica Milenkovic. On unequal error protection ldpc codes based on plotkin-type constructions. *Communications, IEEE Transactions on*, 54(6):994–1005, 2006.

[45] Aleksandr Vasil'evich Kuznetsov and Boris Solomonovich Tsybakov. Coding in a memory with defective cells. *Problemy peredachi informatsii*, 10(2):52–60, 1974.

[46] AV Kuzntsov and AJ Han Vinck. On the general defective channel with informed encoder and capacities of some constrained memories. *Information Theory, IEEE Transactions on*, 40(6):1866–1871, 1994.

[47] Jirapha Limbupasiriporn, Leo Storme, and Peter Vandendriessche. Large weight code words in projective space codes. *Linear Algebra and its Applications*, 437(3):809–816, 2012.

[48] Michael G Luby, Michael Mitzenmacher, Amin Shokrollahi, and Daniel A Spielman. Improved low-density parity-check codes using irregular graphs. *Information Theory, IEEE Transactions on*, 47(2):585–598, 2001.

[49] Michael G Luby, Michael Mitzenmacher, Amin Shokrollahi, and Daniel A Spielman. Improved low-density parity-check codes using irregular graphs. *Information Theory, IEEE Transactions on*, 47(2):585–598, 2001.

[50] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error-correcting codes*, volume 16. Elsevier, 1977.

[51] Yuu Maeda and Haruhiko Kaneko. Error control coding for multilevel cell flash memories using nonbinary low-density parity-check codes. In *Defect and Fault Tolerance in VLSI Systems, 2009. DFT'09. 24th IEEE International Symposium on*, pages 367–375. IEEE, 2009.

[52] Frans Merkx. Womcodes constructed with projective geometries. *Traitment du Signal*, 1:227–231, 1984.

[53] D. E. Muller. Application of boolean algebra to switching circuit design and to error detection. *IRE Transactions Computing*, 3:6–12, 1954.

[54] Hossein Pishro-Nik, Nazanin Rahnavard, and Faramarz Fekri. Nonuniform error correction using low-density parity-check codes. *Information Theory, IEEE Transactions on*, 51(7):2702–2714, 2005.

[55] Charly Poulliat, David Declercq, and Inbar Fijalkow. Enhancement of unequal error protection properties of ldpc codes. *EURASIP Journal on Wireless Communications and Networking*, 2007(3):5, 2007.

[56] Charly Poulliat, Marc Fossorier, and David Declercq. Design of non binary ldpc codes using their binary image: algebraic properties. *optimization*, 5(1):6, 2006.

[57] Charly Poulliat, Marc Fossorier, and David Declercq. Design of regular (2, d/sub c/)-ldpc codes over gf (q) using their binary images. *Communications, IEEE Transactions on*, 56(10):1626–1635, 2008.

[58] John G Proakis and Masoud Salehi. *Fundamentals of communication systems*. Pearson Education India, 2007.

[59] I. S. Reed. A class of multiple-error-correcting codes and the decoding scheme. *IRE Transactions on Information Theory*, 4:38–49, 1954.

[60] Thomas J Richardson, Amin Shokrollahi, and Rüdiger L Urbanke. Design of capacity-approaching irregular low-density parity-check codes. *Information Theory, IEEE Transactions on*, 47(2):619–637, 2001.

[61] Thomas J Richardson, Amin Shokrollahi, and Rüdiger L Urbanke. Design of capacity-approaching irregular low-density parity-check codes. *Information Theory, IEEE Transactions on*, 47(2):619–637, 2001.

[62] Thomas J Richardson and Rüdiger L Urbanke. The capacity of low-density parity-check codes under message-passing decoding. *Information Theory, IEEE Transactions on*, 47(2):599–618, 2001.

[63] Tom Richardson. Error floors of ldpc codes. In *Proceedings of the annual Allerton conference on communication control and computing*, volume 41, pages 1426–1435. The University; 1998, 2003.

[64] Ronald L Rivest and Adi Shamir. How to reuse a ?write-once? memory. *Information and control*, 55(1):1–19, 1982.

[65] Ron Roth. *Introduction to coding theory*. Cambridge University Press, 2006.

[66] Moshe Schwartz and Alexander Vardy. On the stopping distance and the stopping redundancy of codes. *Information Theory, IEEE Transactions on*, 52(3):922–932, 2006.

[67] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[68] Amin Shokrollahi. Capacity-achieving sequences. In *Codes, Systems, and Graphical Models*, pages 153–166. Springer, 2001.

[69] Amin Shokrollahi. Ldpc codes: An introduction. In *Coding, cryptography and combinatorics*, pages 85–110. Springer, 2004.

[70] Amir Shpilka. New constructions of wom codes using the wozencraft ensemble. In *Latin American Symposium on Theoretical Informatics*. Arequipa, Peru, 2012.

[71] Michael Sipser and Daniel A Spielman. Expander codes. *IEEE Transactions on Information Theory*, 42(6):1710–1722, 1996.

[72] Heng Tang, Jun Xu, Shu Lin, and Khaled AS Abdel-Ghaffar. Codes on finite geometries. *Information Theory, IEEE Transactions on*, 51(2):572–596, 2005.

[73] Robert Michael Tanner. A recursive approach to low complexity codes. *Information Theory, IEEE Transactions on*, 27(5):533–547, 1981.

[74] Tadashi Wadayama. Average coset weight distribution of combined ldpc matrix ensembles. *Information Theory, IEEE Transactions on*, 52(11):4856–4866, 2006.

[75] H. Weingarten. *New strategies to overcome 3bpc challenges.* Flash Memory Summit, Santa Clara, 2010.

[76] Yunnan Wu and Anxiao Jiang. Position modulation code for rewriting write-once memories. *Information Theory, IEEE Transactions on*, 57(6):3692–3697, 2011.

[77] Eitan Yaakobi, Laura Grupp, Paul H Siegel, Steven Swanson, and Jack K Wolf. Characterization and error-correcting codes for tlc flash memories. In *Computing, Networking and Communications (ICNC), 2012 International Conference on*, pages 486–491. IEEE, 2012.

[78] Eitan Yaakobi, Scott Kayser, Paul H Siegel, Alexander Vardy, and Jack Keil Wolf. Codes for write-once memories. *Information Theory, IEEE Transactions on*, 58(9):5985–5999, 2012.

[79] Eitan Yaakobi, Jing Ma, Laura Grupp, Paul H Siegel, Steven Swanson, and Jack K Wolf. Error characterization and coding schemes for flash memories. *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*, pages 1856–1860, 2010.

[80] Eitan Yaakobi and Amir Shpilka. High sum-rate three-write and non-binary wom codes. In *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, pages 1386–1390. IEEE, 2012.

[81] Eitan Yaakobi, Paul H Siegel, Alexander Vardy, and Jack K Wolf. Multiple error-correcting wom-codes. volume 58, pages 2220–2230. IEEE, 2012.

[82] Eitan Yaakobi, Alexander Vardy, Paul H Siegel, and Jack K Wolf. Multidimensional flash codes. In *Communication, Control, and Computing, 2008 46th Annual Allerton Conference on*, pages 392–399. IEEE, 2008.

[83] Gilles Zémor. Problèmes combinatoires liés à l'écriture sur des mémoires. 1989.

[84] Gilles Zemor and Gérard Cohen. Error-correcting wom-codes. *Information Theory, IEEE Transactions on*, 37(3):730–734, 1991.

[85] Fan Zhang, Henry D Pfister, and Anxiao Jiang. Ldpc codes for rank modulation in flash memories. In *Information Theory Proceedings (ISIT), 2010 IEEE International Symposium on*, pages 859–863. IEEE, 2010.