Technische Universität München

TUM School of Computation, Information and Technology

Coding and Bounds for Unreliable Data Storage Memories

Haider Abdul Hassan Hadi Al Kim

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr. -Ing.)

genehmigten Dissertation.

Vorsitz:		Prof. Dr. Sebastian Steinhorst
Prüfer*innen der Dissertation:	1.	Prof. Dr. -Ing Antonia Wachter-Zeh
	2.	Prof. Frederic Sala, Ph.D.

Die Dissertation wurde am 17.04.2023 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 20.09.2023 angenommen.

Abstract

EFFECTIVELY storing tremendous amounts of data has become a crucial concern due to the unreliability of storage devices. Memory reliability rapidly decreases with the increase in storage capacity due to various physical impairments. Therefore, developing tailored strategies to make data storage systems ultra-reliable, quick, dense, and inexpensive is essential. Error detection and correction at the data link control (DLC) layer provide reliable communication and fault-tolerant information storage and processing. The DLC is the second telecommunications network layer responsible for detecting and possibly correcting all types of errors.

This dissertation considers code constructions and bounds for *unreliable*, so-called *partially stuck (defective)*, memory cells. Such memory cells can only store partial information as some of their levels cannot be used fully due to, e.g., wear-out. First, we explore the basic principles of channel coding, code families, and the known upper and lower bounds on linear block codes. Second, we identify particular types of data memories and summarize their channel model additive noise and error types that cause reliability issues. Third, we present new constructions that are able to *mask* u partially stuck cells while correcting t substitution errors in the Hamming metric. "Masking" process determines a word whose entries coincide with writable levels at the (partially) stuck cells. Our new constructions coincide and improve upon the required redundancy for masking *partially* stuck cells of known constructions (in the error-free case), and require less redundancy than former works required for masking *fully* stuck cells (which cannot store any information). In our constructions, the encoder outputs a word from an error-correcting code. If this word does not satisfy the partially stuck-at conditions in some positions, the encoder could *deliberately* modify it to fulfill the partially stuck-at constraints. Therefore, in the fourth part, we show that treating some partially stuck cells as erroneous cells can decrease the required redundancy for some parameters. Lastly, we derive Singleton-like, sphere-packing-like, and Gilbert–Varshamov-like (GV-like) bounds. The upper bounds have been derived based on polyalphabetic codes as an encoder's output restricts the values in the partially stuck-at cells; in the other cells, it can attain all values. Polyalphabetic codes use (potentially) distinct alphabets in each coordinate. On the other hand, we employ GV-like lower bound techniques for alphabet size q to confirm the existence of q-ary t-error correcting codes with additional properties. We also study the asymptotic versions of the GV-like bounds. Numerical comparisons state that our constructions match the GV-like bounds for several code parameters, e.g., BCH codes that contain the all-one word by our first construction.

بِسْمِ اللَّـهِ الرَّحْمَـنِ الرَّحِيمِ
(يَرْفَعِ الله الَذينَ آمَنُوا مِنكُمْ وَالَذينَ أُوتُوا الْعِلْمَ دَرَجَاتٍ)
صَدَقَ اللهُ العَلِيُّ العَظيم ـ آية المجادلة:١١

In the name of God, the Gracious, the Merciful
(God raises in ranks those who believed among you and those who have been given knowledge)
Almighty God, the Most High, the Great, has spoken truly - Verse Al-Mujadilah: 11

Acknowledgments

THIS DISSERTATION includes most of my work as a PhD candidate at the Institute for Communications Engineering, TUM School of Computation, Information and Technology, at the Technical University of Munich (TUM), Germany. I want to extend my sincere gratitude to each person who helped me with my research and to have a great time at the institute.

First and foremost, I would like to express my deepest gratitude to my supervisor, Antonia Wachter-Zeh, and my mentor, Sven Puchinger.

I want to thank Antonia for her enthusiastic encouragement to pursue my research in coding theory and its applications, for unending support throughout my Ph.D., and creating such a wonderful, motivating work atmosphere. She encouraged my research and enabled numerous opportunities for me to participate in workshops, lectures, and conferences. Her research network offered innumerable opportunities to collaborate with scientists worldwide.

Sven inspired scientific debates, which significantly impacted my research in many ways. He greatly expanded my perspective on research and directed me to answer emerging questions during my studying.

Additionally, I want to express my appreciation to Ludo Tolhuizen, with whom I had a lot of worthwhile discussions and productive collaborations. Initially, Ludo showed interest in the work direction by reading one of our publications. Then, his helpful criticism always encouraged me to explain my findings more effectively, and finally, together, we extended some results.

The past years were mainly extraordinary because of all my wonderful colleagues, coworkers, and friends at TUM.

At our institute, I am incredibly grateful to all (former) members of the coding group for the scientific discussions: Andreas Lenz, Lukas Holzbaur, Julian Renner, Georg Maringer, Hedongliang Liu, Lorenz Welter, Marvin Xhemrishi, Sabine Pircher, Hugo Sauerbier Couvée, Sebastian Bitzer, Vivian Papadopoulou, Anisha Banerjee, Anna Baumeister, Maximilian Egger, Christoph Hofmeister, Luis Maßny, Anmoal Porwal, Rawad Bitar, Nikita Polyanskii, Violetta Weger, and Yonatan Yehezkeally. Thanks also to Vladimir Sidorenko for discussing and working to develop some research directions related to polyalphabetic codes.

I am also thankful to Chan Kai Jie, Benjamin Koh Yongjie, Sheikh Usman Ali, and Ye Li, undergraduate students who worked with me on their Bachelor's and Master's theses from (TUM, TUM-Asia, and SiT).

My family has always supported me in my decisions throughout my entire life. I would especially like to thank my mother Ahlam Hatif Fatlawi, my father Abdul Hassan Hadi Al Kim, and my siblings Ibrahim, Zainab, Mahmoud, Fatima, Ahmed, Mohammed, and Zahraa for their steadfast support.

Last but certainly not least, I want to thank my own beloved family, which includes my beautiful wife Ruwaida Shaheed, my princess Ayat, my hero Ali, and my adorable baby "the new member" Murtadha, for all that you have given me and continue to provide to me every day.

Haider Al Kim
TUM, November 2023

Contents

Abbreviations

Amag-1	Asymmetric Magnitude-1
CSB	Center Significant Bit
DMC	Discrete Memoryless Channel
DVA	Dynamic Voltage Allocation
ECC	Error-Correcting Codes
BCH	Bose–Ray-Chaudhuri–Hocquenghem
BMD	Bounded Minimum Distance
GRS	Generalized Reed–Solomon
GV	Gilbert–Varshamov
ICI	Inter Cell Interference
LDPC	Low-Density Parity-Check
LSB	Least Significant Bit
Mag-1	Magnitude-1
MDS	Maximum Distance Separable
ML	Maximum Likelihood
MLC	Multi-Level Cell
MSB	Most Significant Bit
NVMs	Non-Volatile Memories
OFDM	Orthogonal Frequency Division Multiplexing
PCC	Partitioned Cyclic Code
PCM	Phase-Change Memory
PDF	Probability Density Function
P/E	Program and Erase Cycles
PMF	Probability Mass Function
Pol	Polyalphabetic
PSMC	Partially Stuck Memory Cell
q-SC	q-ary Symmetric Channel
RM	Reed–Muller
RRE	Reduced Row Echelon
RS	Reed–Solomon
RP	Repetition Code
SMC	Stuck Memory Cell
SPC	Single Parity-Check

TLC	Triple-Level Cell
UMC	Unreachable Memory Cell
WOM	Write Once Memory

Nomenclature

Basics

$[g, f]$	Set of integers $\{i \mid g \leq i \leq f-1\}$
$[f]$	Set of integers $\{i \mid 0 \leq i \leq f-1\}$
$\mathbf{A}$	Matrix
$\mathbf{A}[i, j]$	Element in i-th row and j-th column of $\mathbf{A}$
$\mathbf{A}[i, :]$	i-th row of $\mathbf{A}$
$\mathbf{A}[:, j]$	j-th column of $\mathbf{A}$
$\mathbf{A}^{(\mathcal{I})}$	Matrix $\mathbf{A}$ restricted to the columns indexed by $\mathcal{I}$
$\mathbb{E} \subset [n]$	Erasure pattern subset of $[n]$ positions
$h_q(x)$	q-ary Entropy of random variable x

Matrix and Vector Products

$\mathbf{A} \cdot \mathbf{B}$	(Matrix) product
$\mathbf{A} \times \mathbf{B}$	Cartesian product
$\mathbf{A} \otimes \mathbf{B}$	Kronecker product
$\langle \mathbf{u}, \mathbf{v} \rangle$	Inner product

Codes

$\mathcal{C}$	Code (set of vectors/matrices)
n	Length of a code / number of memory cells
k	Dimension of a code
d	Minimum distance of a code
t	Number of errors
u	Number of partially stuck memory cells
$\boldsymbol{\phi}$	Positions of (partially) stuck memory cells
$n-k$	Redundancy of a code to correct t errors
l	Redundancy of a code to correct u cells

1 Introduction

In a telecommunications network, the OSI model (Open Systems Interconnection), or recently the Internet model, has several intermediate predefined layers in which communication protocols are defined to perform specific tasks. The data link layer, one of the OSI layers, is defined as "Layer 2" and is responsible for *detecting* and possibly *correcting errors* that may occur in the physical layer. Error detection and correction provide *reliable* communication and fault-tolerant information storage and processing [For07].

The demand for reliable storage solutions and, in particular, for *non-volatile memories* (NVMs) such as *flash memories* and *phase-change memories* (PCMs) for different applications is steadily increasing. Non-volatile memory is a memory that stores information even when powered off. These multi-level devices provide permanent storage, a rapidly extendable capacity, faster data access, lower power consumption, and enhanced physical resilience. Thus, they have become the principal hard-disk drive replacement for a range of storage applications [DS16]. Their performance is typically measured using four metrics: lifespan, latency, throughput, and *reliability.*

The amount of time from the point at which data is initially stored to the point at which it is no longer recoverable is referred to as *lifespan.* The period the system takes to complete an activity after receiving a command to retrieve or store data is known as *latency.* The average amount of data the system can store and retrieve in a given amount of time is measured as *throughput.* The probability that data recovered from the system differs from the data initially stored serves as a measure of *reliability.* Flash memory, the most prevalent NVM technology, has been broadly employed in consumer electronics products and industrial electronic systems due to its support for high-throughput and low-latency memory access. A flash device is made up of floating gate transistors that are arranged in a two- or three-dimensional array [DS16]. Every memory cell stores information in the form of charge (voltage levels), which corresponds to a specific digital value. Electrons are programmed to and erased from the floating gate through the oxide layer to regulate the voltage levels [WWCW16]. Two procedures, namely programming/erasing (P/E) and reading, are used to store data and retrieve it from flash-based memories.

That said, a primary issue with flash technology is that its read channel degrades significantly over time, resulting in *unacceptable reliability.* Capturing and releasing charges in flash during programming and erasing causes damage, e.g., *wear-out* in the form of charge *trapping* in the oxide and interface states [CWW14; WCW15; ORS86; MC^{+}09]. The *trap* (the *stuck-at*; see [WY16; Hee83]) prohibits a cell from switching its level, although new charges are injected or removed from this cell. Flash cells generally store one of a few levels–each can be regarded as a symbol over a discrete alphabet of size q. The lowest level generally refers to the entirely erased state, while the highest corresponds to the maximally programmed phase. Due to charge traps, these cells can only access subsets of the q levels. A typical flash unit is a multi-level cell MLC [WWCW16; DS16], which can store four levels, e.g., the values $\{0, 1, 2, 3\}$. Charge trapping at level 1 in an MLC stops the ability to set it back to the lowest level, i.e., the value 0 cannot be modeled anymore. Thus, three out of four values, e.g., $\{1, 2, 3\}$, are the remaining accessible levels. Treating these cells like erasures, i.e., wholly unusable cells, shrinks the storage capacity enormously since, in these cells, subsets of the q levels are still programmable.

Recently developed devices exploit an increased number of cell levels while, at the same time, the physical size of the cells was decreased. For instance, modern flash solutions provide more storage capacity in the form of denser memory devices. Therefore, the resulting increase in physical cell density and signal constellation density amplifies the degradation problem.

On the other hand, the key characteristic of PCM cells is that they can switch between two main states: an *amorphous* state and a *crystalline* state. PCM cells may become *unreliable* (also called *defective* or *stuck*) [GPB09; Hwa^{+}05; Che^{+}22; Pir^{+}04] if they fail in switching their states. This occasionally happens due to the cooling and heating processes of the cells. Thus, cells can only hold a single phase [GPB09; Pir^{+}04]. In multi-level PCM cells, failure may occur at a position in either of extreme states or in the *partially programmable states of crystalline.*

The deterioration, indeed, can be handled at various layers. At the device layer, three-dimensional cell structures in the flash system improve durability [CS11; PBK14; Im^{+}15]. At the system level, *channel codes* can be used since flash-based memories are point-to-point communication systems (see Figure 3.1) and usually require channel codes. The field of channel coding is about developing practical error correction algorithms that introduce controlled, carefully planned redundancy into a data stream to ensure the most *reliable* transmission or storage across a noisy medium. The latter, employing error correction schemes for unreliable memories, is a promising approach to overcoming these physical limitations. It dates back to the 1970s [KT74] and has been extensively researched in [TGKO75; Tsy75b; Tsy75a; Tsy77; BS77; LKD78; KKY78; Hee83; Kuz85; Che85; BV87; Dum87; Dum89; Dum90] under various aspects. In practice, codes like Bose-Chaudhuri-Hocquenghem (BCH) codes [CZW08; CKCH14;

KH14] and low-density parity-check (LDPC) codes [MK09; ZPJ10; Zha$^+$13; Wan$^+$14] add redundancy to protect the stored information. The design of the error-correcting codes ECC significantly impacts the storage system's lifetime, latency, throughput, and reliability. ECC arose from Shannon's seminal 1948 publication [Sha48]. Shannon demonstrated that when the code rate is less than the channel capacity, nearly error-free discrete data transmission is possible over any noisy channel. This statement is now known as the (noisy) channel coding theorem. The physical properties of the channel determine its capacity, and for non-trivial communication channels, it is an active research area. However, due to the non-constructive proof of the channel coding theorem, it is unclear how to construct error-correcting codes that approach the Shannon boundary. With the advent of Hamming's class of codes [Ham50], many articles and textbooks discussing code constructions, their characteristics, and decoding techniques have been appeared and are still currently underway. The Hamming metric is the "appropriate" metric for most data transmission and storage systems, and codes specified in this metric work pretty well in practice.

Since the reliability problems are related to the number of programming and erasing cycles, reducing them by avoiding unnecessarily *erase* states (i.e., preventing the need to return to the *zero* value) is a straightforward solution. Therefore, write once memory (WOM) codes [RS82; Jia07; Yaa$^+$12; GD15], rank modulation [JSB08; MBZ13; QJS13], dynamically voltage allocation (DVA) [WWCW16], and other methods are used to achieve this goal. In other words, the ability to *(re)write* a block of cells without *erasing* is desirable, i.e., minimizing non-mandatory erase cycles. For instance, assuming an MLC cell traps at level 1, writing data using well-designed codes (i.e., their codewords have nonzero components) in the trapped cells means utilizing these cells in their higher remaining levels without demanding mandatory erases.

Therefore, sophisticated coding and signal processing solutions are indispensable to overcome *reliability* issues in non-volatile memories.

1.1 Outline

This dissertation provides coding schemes and bounds for memories of partially defective cells that can only store *partial information*, in which writing on these cells considering the trapped levels and simultaneously correcting substitution errors are proposed. The dissertation is arranged as follows.

Chapter 2 first introduces the notation before providing preliminaries and formal definitions of the channel coding concepts and relevant bounds examined in this dissertation.

Following these preliminary considerations, Chapter 3 reviews non-volatile memories and their channel degradation mechanisms corresponding to error and noise models that make these memories *erroneous* and *defective.* This chapter, thus, introduces our motivation to compromise the influence of these error models in the next chapter.

Chapter 4 is the beginning of the core dissertation. This chapter defines the models of joint errors and partially defective cells examined in this dissertation, and then presents different coding constructions for memories with defects and substitution errors to handle various cases. Despite the fact that the encoding algorithm successfully produces a vector that matches the partial defects as in [WY16], the storing process might fail to present a vector that can be appropriately written to that memory due to substitution errors (i.e., caused by inter-cell interference noise, or other noise disturbance), or the reading process might be unsuccessful [SC19a]. Hence, we consider the problem of combined error correction and *masking* of partially stuck cells. The process of "masking" finds suitable codewords that regard the writable levels in the partially stuck positions.

Compared to the conventional stuck-cell case in [Hee83], we reduce the redundancy necessary for masking, similar to the results in [WY16]. Furthermore, we provide code constructions for *any number of partially stuck cells*; see Table 4.3 for an overview of our constructions and their required redundancies. For the error-free case, where only masking is necessary, our redundancies coincide with those from [WY16] or are even smaller compared to [WY16, Construction 5].

Non-volatile memories actually consist of a vast number of cells, with a (relatively) small number of programmable levels permitted in each. Therefore, they demand channel codes with very long block lengths over a limited range of alphabet sizes. For example, multi-level cell device employs the alphabet $q = 2^2$. Thus, our focus is on long codes over small alphabets, i.e., the code length n is larger than the field size q. Otherwise, one could instead mask by a code of length $n < q$ (by using, e.g., [Sol74]). The final section of Chapter 4 covers codes for *unreachable* memories [GSD14], i.e., a reverse of a partially stuck scenario.

Chapter 5 considers a technique where the encoder *intentionally* introduces errors at some partially stuck positions of a codeword after a first masking step in order to satisfy the stuck-at constraints. The decoder, then, uses part of the error-correcting code capability to correct these purposefully produced errors. We begin with a general proposition of exchanging some error correction possibilities for masking capabilities. We then improve this general proposition for some of our code constructions and give another method for introducing errors by the encoder to fulfill the partially stuck-at restraints. It turns out that treating some partially stuck cells as erroneous cells can increase the stored information for some code parameters.

Chapter 6 is devoted to deriving upper- and lower-like bounds on our constructions, namely Singleton-type, sphere-packing-type (Hamming-type), and Gilbert-Varshamov-type (GV-type) bounds. In our constructions, an encoder output only limits the values in the partially stuck cells; nonetheless, it can achieve all values in the other cells. Consequently, a *polyalphabetic* code can represent the set of all encoder outputs. Error-correcting codes are polyalphabetic if they have (possibly) different alphabets in each entry of their codewords [Sid^{+}05]. Therefore, upper bounds (Singleton-type and sphere-packing-type) on the size of polyalphabetic codes are also upper bounds on the size of partially-stuck-at codes. To this end, we provide two corollaries to state these upper limits for error-correcting and partially-stuck-at masking codes. We finish proposing various GV-type bounds for finite code length based on our coding schemes to show the existence of q-ary t-error correcting codes with additional properties. These lower limits have further been discussed for asymptotic regimes in which the code length tends to infinity, and the number of partially stuck-at cells and the number of random errors grow linearly in the code length.

Chapter 6 *also* contains numerical and analytical comparisons to verify these upper-/lower-type bounds and to evaluate their performance. They have been conducted by comparing the derived bounds based on our code constructions with other trivial codes and known standard limits.

The final section of Chapter 6 briefly summarizes our lower/upper bounds on polyalphabetic codes, taking into account *bounded alphabet sizes*.

Chapter 7 concludes this dissertation.

2 Preliminaries

This chapter introduces and briefly covers the primary principles needed in this dissertation. We begin with Section 2.1, which lists the notations used in the entire thesis. We recall in Section 2.1.1 the vector and matrix multiplications, e.g., the inner product. Then we pursue in Section 2.1.2 to present the polynomials in the finite (extension) field over a single alphabet, which is also covered in the subsequent section (Section 2.2). We briefly introduce in Section 2.3 the theory of error-correcting codes in Hamming metric [Ham50]. Some code families are presented in Section 2.3.4, Section 2.3.5 and Section 2.3.6. The standard upper and lower bounds on linear codes are rephrased in Section 2.4. Section 2.5 summarizes the concept of polyalphabetic codes, i.e., linear codes whose codewords are of coordinates from multiple alphabets. Finally, Section 2.5.1 exhibits the upper limits on these polyalphabetic codes.

2.1 Notation

For a prime power q, let $\mathbb{F}_q$ denote the finite (extension) field of order q, $\mathbb{F}_q^* := \mathbb{F}_q \backslash \{0\}$ be its multiplicative subgroup, and $\mathbb{F}_q[x]$ be the set of all univariate polynomials with coefficients in $\mathbb{F}_q$. The set $\mathbb{F}_q := \{0, 1, \dots, q-1\}$ is also known as *Galois field*. For $g, f \in \mathbb{Z}_{>0}$, denote $[f] = \{0, 1, \dots, f-1\}$ and $[g, f] = \{g, g+1, \dots, f-1\}$. Denote $\mathbb{Z}/q\mathbb{Z}$ for the set of integers modulo q.

As usual, an $[n, k, d]_q$ code (denoted by $\mathcal{C}$) is a linear code over $\mathbb{F}_q$ of length n, dimension k and minimum (Hamming) distance d. The (Hamming) weight $\mathrm{wt}(\boldsymbol{x})$ of a vector $\boldsymbol{x} \in \mathbb{F}_q^n$ equals its number of non-zero entries. The code size (also known as code cardinality) is denoted by M and is given by $M = q^k$. The term "cardinality" defines the number of elements in a given set, and is expressed as $|\,\mathcal{A}\,|$ for any set $\mathcal{A}$. The rate (denoted by R) of the code $\mathcal{C}$ is the ratio of its dimension k and its length n such that $R = {}^{k}\!/_{n}$. The dual code of the code $\mathcal{C}$ is denoted by $\mathcal{C}^\perp$ of the parameters $[n, n-k, d^\perp]_q$, where $d^\perp$ denotes the dual minimum (Hamming) distance. In the asymptotic regime as the code length n tends to infinity, we denote by δ the relative distance which is the ratio of the minimum distance d over the code length n such that $\delta = {}^{d}/_{n}$.

Vectors and matrices are denoted by lowercase and uppercase boldface letters, e.g., $\boldsymbol{a}$ and $\boldsymbol{A}$, respectively, and are indexed starting from 0. For $a, b \in \mathbb{Z}_{>0}$, $\mathrm{RRE}(\boldsymbol{A}^{([b])})$ denotes the reduced row Echelon form of a matrix $\boldsymbol{A}^{([b])}$ that has its columns indexed by the set $[b]$. Denote $\mathbb{F}_q^n$ for the vector space consisting of all length n vectors over $\mathbb{F}_q$, and $\mathbb{F}_q^{a\times b}$ for the matrix space of all $a \times b$ matrices over $\mathbb{F}_q$. For integers $i \in [a]$ and $j \in [b]$ the element in row i and column j is denoted by $A_{i,j}$. To restrict to a given row and to a given column, we write $\boldsymbol{A}[i,:]$ and $\boldsymbol{A}[:,j]$, respectively. $\boldsymbol{A}^{\perp}$ denotes the transpose of $\boldsymbol{A}$. Write $\mathrm{supp}(\boldsymbol{a})$ to denote the set of indices of the non-zero entries (columns) of $\boldsymbol{a}$ and $\mathbf{1}_\ell$ to denote the all-one vector of length ℓ. Note that all calculations are done in the finite field $\mathbb{F}_q$; that is, all calculations are done modulo q.

We fix throughout the thesis a total ordering "$\geq$" of the elements of $\mathbb{F}_q$ such that $a \geq 1 \geq 0$ for all $a \in \mathbb{F}_q \backslash \{0\}$. So 0 is the smallest element in $\mathbb{F}_q$, and 1 is the next smallest element in $\mathbb{F}_q$. We extend the ordering on $\mathbb{F}_q$ to $\mathbb{F}_q^n$: for $\boldsymbol{x} = (x_0, \dots, x_{n-1}) \in \mathbb{F}_q^n$ and $\boldsymbol{y} = (y_0, \dots, y_{n-1}) \in \mathbb{F}_q^n$, we say that $\boldsymbol{x} \geq \boldsymbol{y}$ if and only if $x_i \geq y_i$ for all $i \in [n]$.

In order to simplify notation, we sometimes identify $x \in \mathbb{F}_q$ with the number of field elements not larger than x, that is, with the integer $q - |\{y \in \mathbb{F}_q \mid x \geq y\}|$. The meaning of x will be clear from the context. Figure 3.4 in Section 3.4.3 depicts the two representations that are equivalent in this sense. Finally, we denote the q-ary entropy function by h_q, that is

$$\begin{aligned}&h_q(0) = 0,\ h_q(1) = \log_q(q-1),\ \text{and}\\&h_q(x) = -x\log_q(x) - (1-x)\log_q(1-x) + x\log_q(q-1) \text{ for } 0 < x < 1.\end{aligned}$$

2.1.1 Vector and Matrix Multiplication

Different notations of matrix products can be used in mathematics, particularly in linear algebra. We merely introduce *regular* matrix, vector, and scalar products and restate their relation used in the following chapters.

For $\boldsymbol{A} \in \mathbb{F}_q^{m\times n'}$ and $\boldsymbol{B} \in \mathbb{F}_q^{n'\times n}$, we have:

$$\boldsymbol{A}\cdot\boldsymbol{B} = \begin{pmatrix} A_{0,0} & \dots & A_{0,n'-1} \\ A_{1,0} & \dots & A_{1,n'-1} \\ \vdots & \ddots & \vdots \\ A_{m-1,0} & \dots & A_{m-1,n'-1} \end{pmatrix} \cdot \begin{pmatrix} B_{0,0} & \dots & B_{0,n-1} \\ B_{1,0} & \dots & B_{1,n-1} \\ \vdots & \ddots & \vdots \\ B_{n'-1,0} & \dots & B_{n'-1,n-1} \end{pmatrix}$$

$$= \begin{pmatrix} \langle \boldsymbol{A}[0,:], \boldsymbol{B}[:,0]\rangle & \langle \boldsymbol{A}[1,:], \boldsymbol{B}[:,1]\rangle & \dots & \langle \boldsymbol{A}[0,:], \boldsymbol{B}[:,n-1]\rangle \\ \langle \boldsymbol{A}[1,:], \boldsymbol{B}[:,0]\rangle & \langle \boldsymbol{A}[2,:], \boldsymbol{B}[:,1]\rangle & \dots & \langle \boldsymbol{A}[1,:], \boldsymbol{B}[:,n-1]\rangle \\ \vdots & & \ddots & \vdots \\ \langle \boldsymbol{A}[m-1,:], \boldsymbol{B}[:,0]\rangle & \langle \boldsymbol{A}[m-1,:], \boldsymbol{B}[:,1]\rangle & \dots & \langle \boldsymbol{A}[m-1,:], \boldsymbol{B}[:,n-1]\rangle \end{pmatrix}$$

where

$$\langle \boldsymbol{A}[i,:], \boldsymbol{B}[:,j]\rangle = \sum_{l=0}^{n'-1} A_{i,l} \cdot B_{l,j}$$

is the *inner product* (known also the *dot/scalar product*) between two vectors.

If $\boldsymbol{B} \in \mathbb{F}_q^{1\times n}$, $\boldsymbol{B}$ is a vector, and we denote this vector as $\boldsymbol{b} = (b_0, b_1, \dots, b_{n-1}) \in \mathbb{F}_q^n$.

A scalar is an element in $\mathbb{F}_q$ and denoted by a non-boldface symbol, i.e., $a \in \mathbb{F}_q$. All calculations are carried out in $\mathbb{F}_q$, i.e., taking modulo q for each matrix or vector coordinate. We omit the $\cdot$ symbol whenever it is clear from the context.

2.1.2 Polynomials

Polynomials over the finite (extension) field $\mathbb{F}_q$ and their standard arithmetic operations are briefly explained in the following. For a non-negative integer n, a *polynomial* can be expressed as:

$$a(x) = \sum_{i=0}^{n} a_i x^i, \tag{2.1}$$

where $a_0, a_1, \dots, a_n$ are *elements* in $\mathbb{F}_q$ which are so-called the *coefficients* of the polynomial $a(x)$ [Rot06, page 51].

For a *nonzero polynomial* over $\mathbb{F}_q$ given in (2.1), the *degree* is denoted by $\deg a(x)$ and defined as the largest index i for which $a_i \neq 0$. A nonzero polynomial is called *monic polynomial* if the coefficient a_i of the highest-degree term in the polynomial equals 1. The standard asthmatic operations such as summation and difference, product, and division can be conducted on polynomials (e.g., see [Rot06, page 52]).

2.2 Finite (Extension) Field $\mathbb{F}_q$

The typical literature on finite fields, e.g., [Bla$^+$93; Lid$^+$97], and works on coding theory, e.g., [Ber84; Bla03; Rot06; HP10], provide in-depth analyses of finite fields, their characteristics, and applications.

Let p be *prime*[1] and denote by $\mathbb{F}_p$ the *prime (base) field* of order p. This *finite field* $\mathbb{F}_p$ consists of p elements. A finite field can be defined for *prime power* $q = p^\mu$ and denoted by $\mathbb{F}_q$, where $\mu \geq 1$ is called *extension degree* and p is called its *characteristic*. If the *base* field $\mathbb{F}_p$ is a *subfield* of $\mathbb{F}_q$, i.e., $\mathbb{F}_p \subseteq \mathbb{F}_q$, we say $\mathbb{F}_q$ is an *extension* field of $\mathbb{F}_p$.

Note that $\mathbb{F}_q = \mathbb{F}_p$ for $\mu = 1$; therefore, all properties of extension fields also hold for the special case of prime fields. Hence, addition and multiplication operations in $\mathbb{F}_q$ are equivalent to the corresponding arithmetic operations in $\mathbb{F}_p$ when applied to elements in that base field. An extension field $\mathbb{F}_q$ of $\mathbb{F}_p$ is a vector space over $\mathbb{F}_p$ and is therefore written $\mathbb{F}_{p^\mu}$, where the extension degree μ is the dimension of $\mathbb{F}_p$ such that the bijective mapping $(\mathbb{F}_q = \mathbb{F}_{p^\mu}) \mapsto \mathbb{F}_p^\mu$ holds. This one-to-one mapping is formally called *mapping to base field* (cf. Definition A.1 in Appendix A.1).

2.2.1 Extension Field $\mathbb{F}_q$ by Irreducible Polynomials

Irreducible polynomials over the base field $\mathbb{F}_p$ can alternatively define the extension field $\mathbb{F}_{p^\mu}$, i.e., polynomials in the set $\mathbb{F}_p[x]$ of degree μ for the indeterminate x, whose all coefficients lie in $\mathbb{F}_p$. The standard addition and multiplication, as well as division of polynomials are done in $\mathbb{F}_p$. Corollary 2.11 in [Lid$^+$97] proves that there is at least one such irreducible polynomial for any degree μ. Irreducible polynomials are an analogy to prime[1] numbers for integers.

Before providing a finite extension field using a base field and polynomials, Definition 2.1, Definition 2.2 and Definition 2.3 are needed.

Definition 2.1 (Irreducible Polynomials). *Let $p(x) \in \mathbb{F}_p[x]$ denote a polynomial of degree μ whose coefficients are in $\mathbb{F}_p$. Then $p(x) \in \mathbb{F}_p[x]$ is called irreducible in $\mathbb{F}_p$ if it is not possible to factor it to lower non-zero degree polynomials in $\mathbb{F}_p[x]$.*

Note that different representations of the same extension field $\mathbb{F}_{p^\mu}$ over $\mathbb{F}_p$ are provided by various irreducible polynomials. The reason is that the field $\mathbb{F}_{p^\mu}$ is independent of the choice of an explicit irreducible polynomial $p(x)$ due to the *isomorphic* property of fields of the same size, e.g., see [Lid$^+$97, Theorem 1.78] and [MS77, Theorem 6]. Consequently, the field $\mathbb{F}_{p^\mu}$ is isomorphic to the polynomial ring over $\mathbb{F}_p$ modulo $p(x)$:

$$\mathbb{F}_{p^\mu} \cong \mathbb{F}_p / \langle p(x) \rangle.$$

An irreducible polynomial that fulfills the following conditions is called the *minimal polynomial.*

Definition 2.2 (Minimal Polynomial). *Let $\alpha \in \mathbb{F}_{p^\mu}$ be any element. A monic irreducible polynomial $M(x)$ with coefficients in $\mathbb{F}_p$ such that:*

- *$M(\alpha) = 0$, and*
- *$M(x)$ has the minimal degree*

is called the minimal polynomial of α.

[1]Prime numbers are irreducible integers, and integers can be written as a product of their prime factors. Likewise, polynomials can be written as a product of their irreducible polynomials.

The first condition implies that α is a root of $M(x)$. The second condition means there is no other monic irreducible polynomial of lower degree, e.g., $M'(x)$, satisfying $M'(\alpha) = 0$. The minimal polynomial $M(x)$ is then *irreducible*[2] and *unique*[3]. Hence, $M(x)$ of degree at most μ divides $p(x)$ (cf. Definition 2.1).

Definition 2.3 (Primitive Element)**.** *Let $\alpha \in \mathbb{F}_{p^\mu}$ be an element such that:*

$$\left\{\alpha^0, \alpha^1, \alpha^2, \dots, \alpha^{p^\mu-2}\right\} = \mathbb{F}^*_{p^\mu}, \tag{2.2}$$

and $\alpha^{p^\mu-1} = 1$. Then α is a primitive element of $\mathbb{F}_{p^\mu}$ where all powers of α generate the whole field except 0*.*

There is at least one primitive element in any finite (extension) field [Lid⁺97, page 51]. The minimal polynomial of a *primitive element* α is called the *primitive polynomial.*

The finite extension field then can be expressed as:

$$\mathbb{F}_{p^\mu} := \left\{a(x) = a_0 + a_1x + \cdots + a_{\mu-1}x^{\mu-1} : \, a_j \in \mathbb{F}_p\right\}, \tag{2.3}$$

which is the set of all polynomials $a(x) \in \mathbb{F}_p[x]$ of degree *less* than μ and have coefficients in $\mathbb{F}_p$, with calculations carried out modulo the irreducible polynomial $p(x)$ of degree μ. That is, $x^i \mod p(x) \equiv a(x) \in \mathbb{F}_p[x]$, for all $i = 0, 1, \dots, p^\mu - 2$. There are exactly p^μ such polynomials, which is the size of the extension field $\mathbb{F}_{p^\mu}$ [MS77, Theorem 1]. If $p(x)$ is used to construct $\mathbb{F}_{p^\mu}$ and it satisfies the polynomial in Definition 2.2, then $p(x)$ is the minimal polynomial, e.g., $M(x) := p(x)$.

Remark 2.1. *If the construction of the extension field $\mathbb{F}_{p^\mu}$ has been done using a primitive polynomial (Definition 2.2 and Definition 2.3), then the multiplicative group $\mathbb{F}^*_{p^\mu}$ is a* cyclic *group that has at most $p^\mu - 1$ distinct elements, i.e., any $\alpha \in \mathbb{F}^*_{p^\mu}$ is cyclically repeated such that $1, \alpha^1, \alpha^2, \dots, \alpha^{p^\mu-2}$, with $\alpha^{p^\mu-1} = 1$ [MS77, Thereom 2]. Furthermore, α is a* primitive $(p^\mu - 1)^{th}$ root of unity *as $\alpha^{p^\mu-1} = 1$ where $p^\mu - 1$ is its smallest possible order that satisfies the unity (this property will be needed in our constructions and examples in Chapter 4).*

We see later, in Section 2.3.4, how to construct cyclic and BCH codes using the finite fields and the minimal polynomials.

[2]If $M(x)$ is reducible, then one can write it as $M(x) = M_1(x)M_2(x)$ for any lower degrees polynomials $M_1(x)$ and $M_2(x)$. Then, by the first condition in Definition 2.2 $M(\alpha) = 0$, either $M_1(\alpha) = 0$ or $M_2(\alpha) = 0$, but $M_1(x)$ and $M_2(x)$ of lower degrees which contradicts the second condition.

[3]By [MS77, Theorem 6], all finite fields of order p^μ are isomorphic due to the one-to-one mapping between any two finite fields of order p^μ. [MS77, Theorem 7] shows that due to [MS77, Theorem 6] there is a unique finite field of order p^μ.

2.3 Linear Codes over $\mathbb{F}_q$

A communication system can transmit information from a *source* to a *destination* over a *channel.* The communication may occur in the *time* domain (i.e., by *storing* data at one point in time and *retrieving* it some time later) or *space* domain (i.e., from one location to another) [Rot06, Chapter 1]. Codes were developed to compromise errors on noisy communication lines or storage mediums. According to the most basic description, a code is merely a collection of elements. However, most coding theory literature focuses on *scalar codes* whose elements are called *codewords.* These codewords are vectors of equal lengths over a specific field. The code is *linear* and denoted by $[n, k]_q$ if these codewords span a k-dimensional linear subspace of $\mathbb{F}_q^n$, i.e., the set of all vectors of length n over the *finite fields* $\mathbb{F}_q$. The linearity of a code can be confirmed if any linear combination of codewords in that code is another codeword and the code comprises an all-zero codeword. Throughout this dissertation, we only consider linear codes over $\mathbb{F}_q$, where $q = p^\mu$ for some prime p and $\mu \geq 1$, i.e., q is a prime or a prime power. In case the field size is clear from the context or is not relevant, the linear code is alternatively denoted by $[n, k]$.

If a code is spanned by a k-dimensional space, k linearly independent codewords are its *basis.* Every codeword is then expressible as a unique linear combination of basis vectors. As there are q choices for a scalar multiple of each basis vector, there are, in total, q^k linear combinations. Thus, counting the number of codewords or the so-called *code size* is equivalent to counting the number of linear combinations. For a linear code $\mathcal{C}$ over $\mathbb{F}_q$, the *size* (or commonly known as the *cardinality*) is given by

$$|\,\mathcal{C}\,| = q^k.$$

A matrix $\boldsymbol{G} \in \mathbb{F}_q^{k\times n}$ that includes a basis as its rows is called a *generator matrix* of a linear code. This matrix is a $k \times n$ matrix of rank k.

The code is given by

$$\mathcal{C} = \{\boldsymbol{m} \cdot \boldsymbol{G} \in \mathbb{F}_q^n \,|\, \boldsymbol{m} \in \mathbb{F}_q^k\}. \tag{2.4}$$

The vector $\boldsymbol{m}$ is the *message vector*, and the mapping $\boldsymbol{c} = \boldsymbol{m} \cdot \boldsymbol{G}$ is the message *encoding*, resulting a codeword $\boldsymbol{c}$. Thus, an *encoder* at the transmitter side maps a k-symbol message into an n-symbol codeword in the same alphabet q. Furthermore, the code can be equivalently stated as

$$\mathcal{C} = \{\boldsymbol{c} \,|\, \boldsymbol{c} \in \mathbb{F}_q^n,\, \boldsymbol{c} \cdot \boldsymbol{H}^\top = \boldsymbol{0}\} \tag{2.5}$$

since each linear subspace has a distinct dual space, where the matrix $\boldsymbol{H} \in \mathbb{F}_q^{(n-k)\times n}$ involves as its rows a basis of the dual space of $\mathcal{C}$, called the dual code and denoted by $\mathcal{C}^\perp$. A *decoder* at the receiver side uses $\boldsymbol{c} \cdot \boldsymbol{H}^\top = \boldsymbol{0}$ to infer if the received codeword, say $\hat{\boldsymbol{c}}$, produces the all-zero vector such that $\hat{\boldsymbol{c}} \cdot \boldsymbol{H}^\top = \boldsymbol{0}$. Then, it implies that $\hat{\boldsymbol{c}} = \boldsymbol{c}$ (no errors have inducted). Consequently, the scalar product (cf. Section 2.1.1) of

$\boldsymbol{G} \cdot \boldsymbol{H}^\top = \boldsymbol{0}$ holds since each row of $\boldsymbol{G}$ is a codeword of the code $\mathcal{C}$ whose is generated by $\boldsymbol{G}$.

The dimension $n-k$ of the space spanned by the rows of $\boldsymbol{H}$ is known in the literature of coding theory as the *redundancy* of the code $\mathcal{C}$. The redundancy shows that $n-k$ redundant symbols are needed in a code such that this code is *efficient*, i.e., it can correct many errors while having as large as possible information symbols k. Thus, the amount of redundancy of a code is characterized by its *rate*, which is a code dimension k over its length n and is given by

$$R = \frac{k}{n}.$$

Several metrics decide a code proprieties. We solely consider the *Hamming metric* [Ham50] in this study. The *weight* of a vector $\boldsymbol{c} \in \mathbb{F}_q^n$ under the Hamming metric is the number of non-zero coordinates

$$\mathrm{wt}_H(\boldsymbol{c}) = |\ \mathrm{supp}(\boldsymbol{c})\ | = |\ \{j \mid c_j \neq 0\}\ |.$$

The Hamming distance between two vectors $\boldsymbol{c}, \hat{\boldsymbol{c}} \in \mathbb{F}_q^n$ is defined as the number of differed positions between these two vectors and is officially given by

$$d_H(\boldsymbol{c}, \hat{\boldsymbol{c}}) = |\ \mathrm{supp}(\boldsymbol{c} - \hat{\boldsymbol{c}})\ | = |\ \{j \mid c_j \neq \hat{c}_j\}\ |.$$

Indeed, the *minimum distance* of a linear code, i.e., the smallest number of locations in which any two codewords differ, is a code parameter of a particular interest in coding theory. It is formally defined as

$$d_{min} = \min_{\substack{\boldsymbol{c},\hat{\boldsymbol{c}} \in \mathcal{C} \\ \boldsymbol{c} \neq \hat{\boldsymbol{c}}}} d_H(\boldsymbol{c}, \hat{\boldsymbol{c}}) = \min_{\boldsymbol{c} \in \mathcal{C} \setminus \{0\}} \mathrm{wt}_H(\boldsymbol{c}). \tag{2.6}$$

Note that by the linearity of the code, the second equality holds.

In this work, we simply write *weight* and *distance* and denote them $\mathrm{wt}(\boldsymbol{c})$ and d instead of $\mathrm{wt}_H(\boldsymbol{c})$ and d_H, respectively, since we only consider the Hamming metric. We also write $[n, k, d]_q$ to denote a corresponding linear code over $\mathbb{F}_q$, where d replaces d_{min} to denote the minimum Hamming distance.

The behavior of the distance relative to the code length, also known as the *relative* or *normalized* distance $\delta = {}^{d}/_{n}$, is frequently considered in the asymptotic analysis in the coding theory. The minimal distance should, in general, be as considerable as possible. Research on the bounds of the minimum distance concerning other code parameters is extensive. The bounds of the highest interest for this work are primarily the Singleton bound, the Hamming (sphere-packing) bound, and Gilbert–Varshamov bound, which are found in Sections 2.4.1, 2.4.4, and 2.4.5, respectively.

Definition 2.4 (Subfield Subcodes over $\mathbb{F}_p$)**.** *Given an $[n, k, d]_q$ linear code $\mathcal{C}$ for $q = p^\mu$, where p some prime number and integer $\mu > 1$, then its $\mathbb{F}_p$-subfield subcode,* *denoted by $\mathcal{C}_0$, is defined as follows.*

$$\mathcal{C}_0 := \mathcal{C} \cap \mathbb{F}_p^n = \Big\{ \boldsymbol{c} \mid \boldsymbol{c} \in \mathcal{C},\, c_i \in \mathbb{F}_p \, \forall\, i \in [n] \Big\}.$$

Alternatively, $\mathcal{C}_0$ is given by the $\mathbb{F}_p$ kernel of $\boldsymbol{H} \in \mathbb{F}_{p^\mu}^{(n-k)\times n}$, where $\boldsymbol{H}$ denotes a parity-check matrix of $\mathcal{C}$ such that

$$\mathcal{C}_0 := \mathcal{C} \cap \mathbb{F}_p^n = \Big\{ \boldsymbol{c} \mid \boldsymbol{c} \cdot \boldsymbol{H}^\top = \boldsymbol{0},\, \boldsymbol{c} \in \mathbb{F}_p^n \Big\}.$$

It follows from Definition 2.4 that every codewords of the code $\mathcal{C}_0$ is a codeword of the code $\mathcal{C}$. The parameter d is not (necessarily) the real minimum distance of the code $\mathcal{C}_0$. It is actually known that the distance is larger (see Remark 2.2).

Remark 2.2. *The advantage of codes over the extension fields $\mathbb{F}_{p^\mu}$ for some prime p and extension degree $\mu > 1$ is that they allow designing codes with a considerably large minimum distance for their* constituent *(or* subfield*; cf. Definition 2.4) codes over $\mathbb{F}_p$. For example, nested BCH codes that are defined later in Section 2.3.6 are subfield subcodes of generalized Reed-Solomon codes GRS [Rot06, Chapter 5]. Their minimum distances are bounded from below by the minimum distances of their corresponding GRS codes. At the same time, their dimensions could be as large as these GRS codes.*

The property in Remark 2.2 is useful in our code constructions presented in Chapter 4. In fact, codes over the binary extension fields, i.e., $\mathbb{F}_{2^\mu}$ over $\mathbb{F}_2$ are particularly interesting and considered in this thesis for our code construction in Section 4.5.2.

2.3.1 Error Detection

Error *detection* is a principle that is widely used in practice. It is the event when a receiver becomes aware that an error has emerged, but the correct sent codeword *cannot* be reproduced from the delivered one. The receiver then can either request retransmission or choose to disregard the erroneous codeword.

Figure 2.1 shows the codeword $\boldsymbol{c}$ being sent and $\boldsymbol{y}$ being received. The following theorem demonstrates that the error can always be detected if the distance between $\boldsymbol{c}$ and $\boldsymbol{y}$ is less than the code's minimum distance d.

Theorem 2.1 (Error Detection)**.** *Let $\mathcal{C}$ be a code of minimum distance d such that $\boldsymbol{c} \in \mathcal{C}$ is transmitted (or stored) codeword. Assume the channel adds an error $\boldsymbol{e}$ with $0 \le \mathrm{wt}(\boldsymbol{e}) \le d-1$. Then the decoder can always detect from $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e}$ whether an error aroused or not.*

Proof. If $1 \le \mathrm{wt}(\boldsymbol{e}) \le d-1$, the received (or restored) word $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e}$ cannot be a codeword since the minimum distance between any two codewords is at least d. Hence, the receiver detects that an error transpired. The left case; namely $\mathrm{wt}(\boldsymbol{e}) = 0$, means no error has happened and $\boldsymbol{y} = \boldsymbol{c}$ is a codeword. □

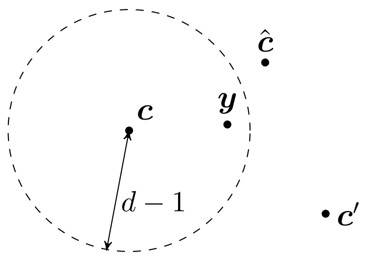


Figure 2.1: Illustration of error detection corresponding to Hamming metric.

2.3.2 Erasure Correction

Linear codes are mainly used to calibrate errors and erasures (or concurrently both) that ensue during transmitting and receiving (or storing and retrieving when regarding storage media) processes across a particular channel. When erasures occur in specified coordinates of a codeword, the corresponding (columns of) symbols in these positions are replaced by an erasure symbol, commonly denoted by $*$ or $\circledast$. An *erasure pattern* $\mathbb{E} \subset [n]$ is correctable if and only if there are not two (or more) codewords that match in all $[n]\backslash\mathbb{E}$ (it reads "the set $[n]$ except its subset $\mathbb{E}$") *surviving* positions (i.e., all locations that have not been erased). The following theorem explains the erasure correction.

Theorem 2.2 (Erasure Correction)**.** *Let $\mathcal{C}$ be a code of minimum distance d such that $\boldsymbol{c} \in \mathcal{C}$ is transmitted (or stored) codeword. Assume the channel erases at most $d-1$ symbols and their locations are known given in the set $\mathbb{E}$. Then the decoder can always correct the erased coordinates.*

Proof. Since any two codewords differ in at least d locations, a code $\mathcal{C}$ with a minimum distance d ensures that any combination of up to $|\mathbb{E}| \leq d-1$ erasures can be corrected, regardless of their positions in the codeword. Hence, fixing $d-1$ positions in all the codewords reveals that any two words still differ by at least one symbol. $\square$

The former condition is not a key prerequisite for *correctability*; it is merely a sufficient one. An erasure pattern can typically be corrected if the mapping from the message $\boldsymbol{m} \in \mathbb{F}_q^k$ to the surviving coordinates, such that $\boldsymbol{c}^{(j)} = \boldsymbol{m} \cdot \boldsymbol{G}^{(j)}$ for $j \in [n]\backslash\mathbb{E}$, is still injective, where the code $\mathcal{C}$ is generated by the matrix $\boldsymbol{G}$. It is obvious that the prior case is true if and only if the *submatrix* of $\boldsymbol{G}$ whose columns are labeled by $j \in [n]\backslash\mathbb{E}$ is of a full rank k. To this end, the least sets of columns for which this holds are called the *information sets* of the code $\mathcal{C}$, i.e., the sets $\mathcal{I} \in [n]$ such that $|\mathcal{I}| = k$ and $\operatorname{rank}(\boldsymbol{G}^{(\mathcal{I})}) = k$. Therefore, it is sufficient to demonstrate that the complement of a particular erasure pattern contains an information set of the code in order to show the ability of a code to correct that specific erasure pattern [HPYW21, Proposition 2]. The dual code $\mathcal{C}^{\perp}$ and the parity-check matrix $\boldsymbol{H}$ can be used as another option to provide a similar condition for an erasure pattern's correctability. Uncomplicated linear algebra arguments make it simple to confirm that the erased positions can only be corrected if they include the dual code's information set; that is, $\operatorname{rank}(\boldsymbol{H}^{(\mathbb{E})}) = n - k$.

We have introduced the erasure correction principle since our work provides a similar concept of known-location faulty patterns (cf. Section 3.4.3) that can be corresponded during the *encoding* procedure. Contrary to the defective patterns in our scenarios considered in Chapter 4, erasure patterns can be treated at the *decoder* regarding the prior argument on correctability.

2.3.3 Error Correction and Decoder Sorts

In comparison to erasures, the fundamental challenge when examining errors is that their positions are not typically known. Suppose a codeword $\boldsymbol{c}$ of a q-ary code $\mathcal{C}$ is transmitted (or stored) over a channel (or a storage medium) that poses errors, i.e., flips some coordinates of the codeword. The channel output is the *received* (or *retrieved*) word which is of the form $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e}$ and $\operatorname{supp}(\boldsymbol{e})$ indicates the set of *error locations*. The decoder, in this regard, is primarily willing to *decode* this received (restored) word, i.e., reconstructing the codeword that is most likely to be the transmitted (stored) codeword. Since the decoder performs a non-trivial decoding task, three types of decoders are distinguished: maximum likelihood (ML) decoder, bounded minimum distance (BMD) decoder, and list decoder. We direct the interested readers to check the maximum likelihood (ML) decoder in [Rot06, Chapter 1, page 8]. We explain the bounded minimum distance (BMD) decoder in the following, and for completeness, we subsequently define the list decoder since it is a generalization of BMD.

Bounded Minimum Distance (BMD) Decoder

It is also called *unique decoder* and is used to *guarantee* error correction. The BMD decoder requires a stricter maximum error weight constraint compared to error detection. The following theorem explains the error correction by a BMD decoder.

Theorem 2.3 (Error Correction (BMD))**.** *Let $\mathcal{C}$ be a code of minimum distance d such that $\boldsymbol{c} \in \mathcal{C}$ is transmitted (or stored) codeword. Assume the channel adds an error $\boldsymbol{e}$ with $0 \leq \operatorname{wt}(\boldsymbol{e}) \leq \lfloor \frac{d-1}{2} \rfloor$. Then the decoder can always correct the errors in the received (or restored) word $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e}$.*

Proof. Since any two codewords differ in at least d locations, a code $\mathcal{C}$ with a minimum distance d ensures that any error of weight $0 \leq \operatorname{wt}(\boldsymbol{e}) \leq \lfloor \frac{d-1}{2} \rfloor$ can be corrected. The fact that decoding spheres of radius $\lfloor \frac{d-1}{2} \rfloor$ surrounding all codewords do not intersect, as shown in Figure 2.2, guarantees unique correctability by finding the center of the decoding sphere in which the received $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e}$ belongs. □

A BMD decoder returns either a unique codeword or an empty set; in the latter, it declares a decoding failure. Figure 2.2 depicts a BMD decoder in which the received

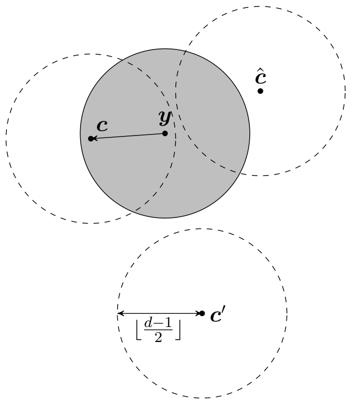


Figure 2.2: BMD decoder for q-ary symmetric channel corresponding to Hamming metric.

word $\boldsymbol{y}$ is mapped to the codeword $\boldsymbol{c}$ which is the only one that lies inside the decoding radius. Many classes of linear codes (in the corresponding metric) have effective BMD decoders. However, it should be noted that this is typically not an easy problem, as it is unclear how to effectively decode a random linear code up to its *unique decoding radius*. Although the BMD decoder offers the benefit of a decoding guarantee, it can only be used for half the minimum distance. Therefore, *list decoding* presented in the following section is a one way to fix this.

List Decoder

A list decoder aims to deliver a *list* of all codewords that fall inside an *expanded decoding radius*, which goes beyond the unique decoding radius. Let $r > \lfloor\frac{d-1}{2}\rfloor$ be an extended decoding radius; then, there are overlaps in the Hamming spheres. The concept of list decoding can be seen as a generalization of unique decoding, i.e., the list has an individual codeword, and its size equals one. Consequently, a decoding failure is declared if the output is an empty set as well. Figure 2.3 shows a drawing of a sphere of radius r around the received word $\boldsymbol{y}$ such that all codewords that lie in this sphere are an output of a list decoder, e.g., $\boldsymbol{y}$ is mapped to a list of size two that contains the codewords $\boldsymbol{c}$ and $\hat{\boldsymbol{c}}$. Note that the larger the radius r, the higher the complexity, as the list size can grow exponentially in the code length, which makes the list decoder practically *infeasible*. This means that the radius r cannot be arbitrarily large. We shall see later in Section 2.4.4 how to find the number of words in a sphere of a given radius, i.e., the *volume* of a sphere.

In this dissertation, we only consider *unique decoding (BMD decoder)* corresponding to Hamming metric [Ham50] for q-ary code (cf. Section 2.3.3) to guarantee the capability of correcting any error vector $\boldsymbol{e}$ of weight $0 \leq \mathrm{wt}(\boldsymbol{e}) \leq \lfloor\frac{d-1}{2}\rfloor$ in our code constructions given in Chapter 4. Nevertheless, we shall see in Appendix A.1 a potential extensions of this work suggested upon *array codes* and a *list decoder*.

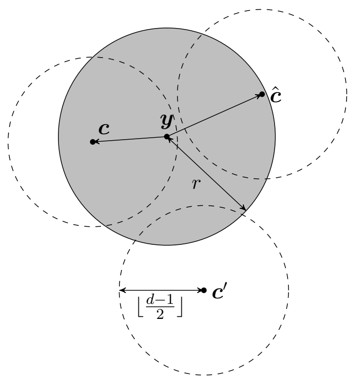


Figure 2.3: List decoder for q-ary symmetric channel corresponding to Hamming metric.

2.3.4 Cyclic Codes

Linear codes over finite fields are *cyclic codes* if they fulfill the following definition.

Definition 2.5 (Cyclic Codes)**.** *Let $\mathcal{C}$ be a linear code over $\mathbb{F}_q$. If any cyclic shift of a codeword $\boldsymbol{c} = (c_0, c_1, \dots, c_{n-1}) \in \mathcal{C}$ is again a codeword in $\mathcal{C}$ such that*

$$(c_0, c_1, \dots, c_{n-1}) \in \mathcal{C} \Longrightarrow (c_{n-1}, c_0, \dots, c_{n-2}) \in \mathcal{C},$$

then $\mathcal{C}$ is cyclic.

Definition 2.5 indicates unequivocally that a shift of i positions for a codeword components gives another codeword. For cyclic codes, polynomial representations for codewords are used to simplify the notation. Each codeword $\boldsymbol{c} = (c_0, c_1, \dots, c_{n-1}) \in \mathbb{F}_q^n$ is associated with a polynomial $c(x) = c_0 + c_1x + c_2x^2, \dots, c_{n-1}x^{n-1} \in \mathbb{F}_q[x]$ of degree $\deg c(x) \leq n-1$, where its coefficients $c_0, c_1, \dots, c_{n-1} \in \mathbb{F}_q$. Consequently, the cyclic shift then corresponds to:

$$c_{n-1} + c_0x + \dots + c_{n-2}x^{n-1} = x \cdot c(x) - c_{n-1} \cdot (x^n - 1) \equiv x \cdot c(x) \mod (x^n - 1).$$

Thus, we say a linear code is cyclic if and only if

$$c(x) \in \mathcal{C} \Rightarrow x \cdot c(x) \mod (x^n - 1) \in \mathcal{C},$$

and for every $p(x) \in \mathbb{F}_q[x]$,

$$c(x) \in \mathcal{C} \Rightarrow p(x) \cdot c(x) \mod (x^n - 1) \in \mathcal{C}.$$

We define the polynomials: *generator* and *parity-check* polynomials which correspond to the generator and parity-check matrices, respectively (cf. Section 2.3). A generator polynomial is a unique monic polynomial of the smallest degree, denoted by $g(x)$.

Proposition 2.1. *Let code $\mathcal{C} \subseteq \mathbb{F}_q^n$ fulfill Definition 2.5 and of the parameters $[n, k > 0, d]_q$. Any codeword $c(x) \in \mathbb{F}_q[x]$ of deg $c(x) \leq n-1$ of the code $\mathcal{C}$ must satisfy:*

$$c(x) \in \mathcal{C} \iff g(x) | c(x),$$

and can be expressed as

$$c(x) = m(x) \cdot g(x), \tag{2.7}$$

for some message polynomial $m(x)$ of deg $m(x) \leq k-1$ and generator polynomial $g(x)$ of deg $g(x) = n-k$.

Proof. We observe that $g(x)$ is a codeword in the code $\mathcal{C}$ since surely $g(x)$ divides itself. Now since $g(x)|c(x)$, we write $c(x) = m(x) \cdot g(x) + r(x)$ with deg $r(x) <$ deg $g(x)$. By the linearity of the code $\mathcal{C}$, $r(x) = c(x) - m(x) \cdot g(x) \in \mathcal{C}$. Still, by the requirement on the generator polynomial, $g(x) \in \mathcal{C}$ is the *smallest* degree nonzero polynomial. We conclude that $r(x) = 0$ since its degree is less than $g(x)$.

Additionally, we observe that $x^n - 1 = h(x) \cdot g(x) + r(x)$ with deg $r(x) <$ deg $g(x)$; therefore, $r(x) \equiv -h(x)g(x) \mod (x^n - 1)$. Again as $r(x) = 0$, it means $g(x)|(x^n - 1)$, which is an essential property of a generator polynomial to produce a cyclic code [Rot06, page 246]. □

Concerning a parity-check matrix in a linear code, in this context, we define a parity-check polynomial as

$$h(x) := \frac{x^n - 1}{g(x)}, \tag{2.8}$$

where $h(x)$ is of degree k. The equivalent definition for cyclic code is then

$$c(x) \cdot h(x) = 0 \mod x^n - 1.$$

Since $h(x) \cdot g(x) = x^n - 1$ by (2.8), we can always *split* nontrivial[4] cyclic codes such that $1 \leq k \leq n-1$ to define the degrees of $g(x)$ and $h(x)$.

Partitioned Cyclic Code

The concept of a *partitioned* cyclic code introduced in [Hee83] takes advantage of the degree distribution between $g(x)$ and $h(x)$.

Definition 2.6 (Partitioned Cyclic Code)**.** *Let $\mathcal{C}$ be a cyclic code fulfilling Definition 2.5 that has $g(x)$ of degree deg $g(x) = n-k$ and $h(x)$ of degree deg $h(x) = k$. Assume there is another generator polynomial $g_0(x)$ of degree deg $g_0(x) = l$ that has the following properties:*

- *$g_0(x)|h(x)$ for $l \leq k$, and*

[4]Trivial cyclic codes are linear codes with $g(x) = 1$ or $g(x) = x^n - 1$.

- *$g_0(x)$ shares no common roots with $g(x)$.*

Then $\mathcal{C}$ could be partitioned such that any word $c(x) \in \mathcal{C}$ can be expressed as

$$c(x) = c_1(x) + c_0(x) = m(x)g(x) + m_0(x)g_0(x), \tag{2.9}$$

where the message polynomial $m(x)$ is of degree deg $m(x) \leq i - j$ and the message polynomial $m_0(x)$ is of degree deg $m_0(x) \leq j$ for $0 \leq j \leq i \leq k-1$.

Generator and Parity-check Matrices for Cyclic Codes

Generator and parity-check matrices can be obtained by their corresponding generator and parity-check polynomials, receptively, as follows. We write $g(x) = g_0 + g_1x + g_2x^2 + \cdots + g_{n-k}x^{n-k}$ so that its coefficients are $g_0, g_1, \ldots, g_{n-k}$ can be used to define a generator matrix $\boldsymbol{G}$ of the dimensions $k \times n$ for a cyclic code in the following form:

$$\boldsymbol{G} = \begin{pmatrix} g_0 & g_1 & \cdots & g_{n-k} & & & & \\ & g_0 & g_1 & & \cdots & g_{n-k} & & 0 \\ 0 & & \ddots & & \ddots & & \ddots & \ddots \\ & & & g_0 & & g_1 & \cdots & g_{n-k} \end{pmatrix} \in \mathbb{F}_q^{k \times n}. \tag{2.10}$$

Similarly, we write $h(x) = h_0 + h_1x + h_2x^2 + \cdots + h_kx^k$, then the related parity-check matrix $\boldsymbol{H}$ of the dimensions $(n-k) \times n$ can be defined in the following form:

$$\boldsymbol{H} = \begin{pmatrix} h_k & h_{k-1} & \ldots & h_0 & & & & \\ & h_k & h_{k-1} & \ldots & & h_0 & & 0 \\ 0 & & \ddots & \ddots & & \ddots & \ddots & \\ & & & h_k & h_{k-1} & \ldots & h_0 & \end{pmatrix} \in \mathbb{F}_q^{(n-k) \times n}. \tag{2.11}$$

Thus, a codeword polynomial, namely $c(x)$ from (2.7), can be written in a vector form such that

$$\boldsymbol{c} \in \mathcal{C} := \boldsymbol{m} \cdot \boldsymbol{G} = (m_0, m_1, \ldots, m_{k-1}) \cdot \begin{pmatrix} g_0 & g_1 & \cdots & g_{n-k} & & & & \\ & g_0 & g_1 & & \cdots & g_{n-k} & & 0 \\ 0 & & \ddots & & \ddots & & \ddots & \ddots \\ & & & g_0 & & g_1 & \cdots & g_{n-k} \end{pmatrix} \in \mathbb{F}_q^n,$$

where $\boldsymbol{m} = (m_0, m_1, \ldots, m_{k-1}) \in \mathbb{F}_q^k$ corresponds to

$$m(x) = m_0 + m_1x + m_2x^2, \ldots, m_{k-1}x^{k-1} \in \mathbb{F}_q[x],$$

which is called the *message polynomial* of degree $\deg m(x) = k - 1$ with coefficients

$$m_0, m_1, \ldots, m_{k-1} \in \mathbb{F}_q.$$

It also holds that $\boldsymbol{c} \cdot \boldsymbol{H}^\top = \boldsymbol{0}$ by (2.5) for $\boldsymbol{H}$ defined by (2.11). It is known that a dual code $\mathcal{C}^\perp$ of a cyclic code $\mathcal{C}$ is also a *cyclic* code over $\mathbb{F}_q$ of length n, dimension $n - k$, and some dual minimum distance $d^\perp$ [Rot06, Corollary 8.4]. Then the dual code of an $[n, k, d]_q$ cyclic code $\mathcal{C}$ is an $[n, n-k, d^\perp]_q$ cyclic code with a generator polynomial

$$g^\perp(x) := \frac{x^k \cdot h(x^{-1})}{h(0)}, \tag{2.12}$$

where the degree of $\deg g^\perp(x) = n - k^\perp = k$ and the degree of its parity-check polynomial $h^\perp(x)$ $\deg h^\perp(x) = n - k$.

So far, we have regarded the length n and the dimension k of a cyclic code $\mathcal{C}$. The minimum distance d of $\mathcal{C}$ is as defined by (2.6) and can be bounded from below by the *Bose-Ray-Chaudhuri-Hocquenghem* (BCH) bound (introduced in Section 2.3.6) [MS77, Chapter 9], or more involved bounds such as the Hartmann-Tzeng bound [HT72] or the Roos bound [Roo79].

2.3.5 q-ary Repetition and Single parity-check Codes

Repetition codes (RP) of the parameters $[n, 1, n]_q$ are cyclic codes of the generator and parity-check polynomials, respectively, as follows:

$$g(x) := 1 + x + x^2 + \cdots + x^{n-1}, \tag{2.13}$$

$$h(x) = \frac{x^n - 1}{g(x)} = x - q + 1. \tag{2.14}$$

Then, their dual codes are single parity-check codes (SPC) of the parameters $[n, n-1, 2]_q$, which are also cyclic codes of the following generator and parity-check polynomials (respectively)

$$g^\perp(x) := \frac{x^k \cdot h(x^{-1})}{h(0)} = x - q + 1, \tag{2.15}$$

$$h^\perp(x) = 1 + x + x^2 + \cdots + x^{n-1}. \tag{2.16}$$

Definition 2.6 explains the degree splitting between $g(x)$ and $h(x)$. Notice that taking $k = n - 1$ for $h(x)$ in Definition 2.6 gives (2.16), and accordingly, $g(x)$ coincides (2.15). We shall benefit from this spacial case of partitioned cyclic code in Construction 4.3, located in Chapter 4. We will also see in Chapter 4 the usability and uniqueness of these code classes, e.g., looking ahead to Example 4.2.

2.3.6 Bose-Ray-Chaudhuri-Hocquenghem (BCH) Codes

Before defining BCH codes [MS77, Chapter 9], a particular type of cyclic code, we introduce the notation of *cyclotomic cosets*. For that, we first restate Fermat's theorem [MS77, Corollary 3, page 96].

Corollary 2.1 (Fermat's Theorem). *Let α be any element over $\mathbb{F}_{p^\mu}$ of order p^μ. Then α satisfies the identity such that*

$$\alpha^{p^\mu} = \alpha,$$

or is equivalently a root of the equation:

$$x^{p^\mu} = x.$$

Hence, the following holds:

$$x^{p^\mu} - x = \prod_{\alpha \in \mathbb{F}_{p^\mu}} (x - \alpha). \tag{2.17}$$

Corollary 2.1 implies that $x^{p^\mu} - x = 0$, or alternatively, dividing both sides by x such that $x^{p^\mu - 1} - 1 = 0$ considers all the nonzero α in $\mathbb{F}_{p^\mu}$, i.e., the multiplicative group $\mathbb{F}^*_{p^\mu}$ (cf. Definition 2.3). Indeed, by (2.17) $x^{p^\mu-1} - 1$ can be factored in $\mathbb{F}_{p^\mu}[x]$ to its degree-one polynomials and rewritten as follows:

$$x^{p^\mu-1} - 1 = (x-1) \cdot (x - \alpha) \cdot (x - \alpha^2) \cdot \cdots \cdot (x - \alpha^{p^\mu - 2}). \tag{2.18}$$

Now, we look at the factorization of $x^{p^\mu-1} - 1$, but over the base field in $\mathbb{F}_p[x]$. Suppose $M(x) \in \mathbb{F}_p[x]$ is a monic irreducible factor of $x^{p^\mu-1} - 1$ that has some root α^a (cf. Definition 2.2). By the Frobenius automorphism [FT91, page 144] for any $\alpha \in \mathbb{F}_{p^\mu}$ it holds that $\alpha \mapsto \alpha^p$ (it reads "α maps to α^p"), then $M(x)$ has also roots

$$\alpha^{a \cdot p}, \alpha^{a \cdot p^2}, \ldots, \alpha^{a \cdot p^{n_a}} = \alpha^a,$$

for some integer n_a that taking its modulo $p^\mu - 1$ gives back α^a. Hence, $M(x)$ has *all* α^b for $b \in \{a, a \cdot p, a \cdot p^2, \ldots, a \cdot p^{n_a - 1}\} := \mathcal{J}$ as roots. Consequently, the product

$$\prod_{b \in \mathcal{J}} (x - \alpha^b)$$

that lives in $\mathbb{F}_p[x]$ [Rot06, page 220] divides $M(x)$. Since $M(x)$ is irreducible then indeed

$$M(x) = \prod_{b \in \mathcal{J}} (x - \alpha^b). \tag{2.19}$$

Considering (2.18), some of the powers of α fall into disjoint sets, and therefore they have the same minimal polynomial $M(x)$ defined by (2.19). While minimal polynomials are defined over $\mathbb{F}_{p^\mu}$, they are in fact polynomials in the base field $\mathbb{F}_p$ [Rot06, Proposition 7.3]. To this end, multiplying by the prime p divides the set of integers modulo $p^\mu - 1$ into cyclotomic cosets, and each one defines its corresponding minimal polynomial.

Definition 2.7 (Cyclotomic Cosets). *Let integer n have* $\gcd(n, p^\mu) = 1$. *Let μ be the smallest integers such that n divides $p^\mu - 1$. A cyclotomic coset M_a with respect to n is given by:*

$$M_a := \left\{a \cdot p^j \mod n,\ \forall j = 0, 1, \cdots, n_a - 1\right\}, \tag{2.20}$$

where n_a is the smallest integer such that $a \cdot p^{n_a} \equiv a \mod n$.

Let $\alpha \in \mathbb{F}_{p^\mu}$ be a primitive n^{th} root of unity in $\mathbb{F}_{p^\mu}$, i.e., $\alpha^n = 1$ and n its smallest possible order that satisfies the unity (refer also to Definition 2.3 and Remark 2.1). The minimal polynomial (cf. Definition 2.2) of an element α^a is given by:

$$M^{(a)}(x) := \prod_{b \in M_a} (x - \alpha^b). \tag{2.21}$$

We define a *BCH code* as the following.

Definition 2.8 (BCH Codes). *Let a cyclic code of length n, dimension k, and minimum distance d be an $[n, k, d]_p$ code $\mathcal{C}$. It has a* generator polynomial *$g(x)$ of degree deg $g(x) = n - k$ with roots in $\mathbb{F}_{p^\mu}$, where n divides $p^\mu - 1$.*

The defining set D_c of the code $\mathcal{C}$ is the set containing the indices b of the root α^b of the generator polynomial $g(x)$ such that

$$D_c := \{b : g(\alpha^b) = 0\} = M_{a_1} \cup M_{a_2} \cup M_{a_3} \cdots \cup M_{a_w}. \tag{2.22}$$

Then, $g(x) \in \mathbb{F}_p[x]$ is thus given by

$$g(x) = \prod_{a \in D_c} (x - \alpha^a) = \prod_{b=1}^{w} M^{(a_b)}(x). \tag{2.23}$$

Hence, deg $g(x) = n - k = |D_c|$ which is defined as the redundancy *of the code $\mathcal{C}$. For any cyclic code, there is a* parity-check polynomial *that is given by:*

$$h(x) = \frac{(x^n - 1)}{g(x)} = \prod_{a \in [n] \setminus D_c} (x - \alpha^a). \tag{2.24}$$

Possible lengths n follow from the definition of the defining set D_c. Some lengths are not feasible when creating a BCH code, specifically the length needs to be co-prime with p, which is stated in the following lemma.

Lemma 2.1. *$n|(p^\mu - 1)$ implies that gcd$(n, p^\mu) = 1$.*

Proof. $n \mid (p^\mu - 1)$ means

$$\exists\, k \in \mathbb{N}, \quad \text{such that} \qquad k \cdot n = p^\mu - 1.$$

Furthermore, suppose now $\gcd(n, p) = c$ with $c \in \mathbb{N}$, then it follows

$$\begin{aligned}&\exists\, a, b \in \mathbb{N}, \quad \text{such that}\\ &a \cdot c = p,\\ &b \cdot c = n.\end{aligned}$$

Substituting n and p with this, we get

$$\begin{aligned}k \cdot bc &= (ac)^\mu - 1\\ k \cdot bc - a^\mu c^\mu &= -1\\ c\left(-k \cdot b + a^\mu c^{\mu-1}\right) &= 1.\end{aligned}$$

Since $k, a, b, c \in \mathbb{N}$ it needs to hold that $c = 1$. □

By Lemma 2.1, only certain combinations of p, μ, and n enable the construction of BCH codes. The indices $1, \ldots, w$ that define D_c in (2.22) allow designing the dimensions k of BCH codes since $k = n - |D_c|$, where $|D_c| = \sum_{b=1}^{w} |M_{a_b}|$. However, not all values from $1, \ldots, n$ are possible selections for k since $|D_c|$ is a combination of one or more cyclotomic cosets, and consequently, they define one or more minimal polynomials (of possibly different degrees) to provide $g(x)$.

Furthermore, D_c permits to design the code minimum distance d since any consecutive elements in D_c plus one gives a bound on d. Let $\alpha^b, \alpha^{b+1}, \ldots, \alpha^{b+D-2}$ be roots belonging to the set of roots of $\mathcal{C}$ defined by (2.22) for some integers b and $D \geq 2$. Then the *designed* minimum distance of the code $\mathcal{C}$ is $d \geq D$ [Rot06, Proposition 8.7]. Note that if $n = p^\mu - 1$, the BCH code is called a *primitive* BCH code (cf. Definition 2.3).

BCH codes can be derived from an underlying Reed-Solomon (RS) codes [Rot06, Chapter 5] of a larger field, i.e., over a *splitting* field $\mathbb{F}_{p^h}$ for integer $h \geq 1$. Denote $\mathcal{C}_{RS} \subseteq \mathbb{F}_{p^\mu}$ the underlying RS code for prime p and some $\mu > 1$. Then by Definition 2.4, there is a BCH code $\mathcal{C} \subseteq \mathbb{F}_{p^h}$ defined as

$$\mathcal{C} := \mathcal{C}_{RS} \cap \mathbb{F}_{p^h}^n.$$

BCH codes over $\mathbb{F}_{p^h}$ of degrees $1 \leq h < \mu$ are known as *nested* BCH codes. Then any codeword $c(x)$ of the code $\mathcal{C}_{RS}$ with coefficients in $\mathbb{F}_{p^\mu}$ has to be a multiple of one or more minimal polynomials in $\mathbb{F}_p[x]$ (see (2.3)) while simultaneously being a multiple of the generator polynomial of the code $\mathcal{C}$. Equivalently, for BCH codes, the defining set (cf. Definition 2.8) of the code $\mathcal{C}$ is a subset of the defining set of $\mathcal{C}_{RS}$. Example 4.5 in Chapter 5 shows a nested BCH code.

Since BCH codes (cf. Section 2.3.6) and SPC codes (cf. Section 2.3.5) are cyclic codes themselves, *partitioned* BCH and SPC codes are also possible for some achievable dimensions $1 < k \leq n-1$ (check Definition 2.6).

Observe also that RP and SPC codes in Section 2.3.5 are BCH codes, e.g., (2.13) and (2.14) are maximal and minimal degree versions of (2.23) and (2.24), in which $\deg g(x) = n-1$ and $\deg h(x) = 1$, respectively.

Throughout this thesis, we use q-ary cyclic codes, in particular BCH codes, to describe our coding methods, show some examples and provide code constructions relying on these code classes.

2.4 Bounds on the Cardinality and Minimum Distance

We have defined linear codes and their parameters. In this section, we establish conditions on these parameters. These conditions relate bounds between the code length n, its dimension k (or its size q^k), its rate $R = {}^{k}\!/\!_{n}$, and its minimum distance d. Some of these bounds imply *necessary conditions* on the values n, k, d and q, i.e., the bounds in Section 2.4.1 and Section 2.4.4. We also state the condition of the existence of codes, which is the bound in Section 2.4.5, whenever their parameters n, k, d, and q fulfill an inevitable inequality. Additional bounds are included in Section 2.4.2 and Section 2.4.3, which are upper bounds for codes that encompass an n-weight codeword in their codebooks[5].

We also give asymptotic statements where the code length $n \to \infty$. Asymptotic bounds can be defined such that the asymptotic rate of a code corresponds to its relative minimum distance (denoted by $\delta = {}^{d}\!/\!_{n}$).

Codes are classified as if they attain each of these bounds or not. The chapters [Rot06, Chapter 4] and [MS77, Chapter 17] are two of many references to these (asymptotic) bounds on code parameters. The references provide problems and examples of which code families achieve these limits.

In this dissertation, we explore the standard upper and lower bounds to provide reference limits as a direct comparison with our derived bounds in Chapter 6.

2.4.1 The Singleton Bound

An *upper* bound on the *size* (*cardinality*) or the *minimum distance* for any $[n, k, d]$ linear code over $\mathbb{F}_q$ is the *Singleton* bound (cf. [MS77, Chapter 17]). It states that for any $[n, k, d]$ code

$$d \leq n - k + 1, \tag{2.25}$$

[5]A codebook is the collection of all codewords belonging to a code.

where n is the length of a code, k is its dimension and d is its minimum distance. The code size q^k is then bounded from above as follows:

$$q^k \leq q^{n-d+1}. \tag{2.26}$$

Codes satisfying (2.25) with equality are referred to *maximum distance separable* (MDS) codes. It is known that MDS codes exist for any choices of length n and dimension k, but only if the field size is large enough, generalized Reed–Solomon (GRS) [MS77, Chapter 10] codes are MDS for any prime $q \geq n-1$.

We do not consider in this work $q \geq n-1$ as our code constructions (see Chapter 4) are for linear codes with relatively small alphabet sizes, i.e., $q < n$. The reason is that we design coding algorithms for memories of a vast number of n cells with limited programmable q levels. Nevertheless, MDS codes are essential in our study for fully comprehension and comparisons (see [Sol74]) and since their subfield subcodes (regarding Remark 2.2), e.g., BCH codes (cf. Section 2.3.6) with $q < n$ or alternant codes [MS77, Chapter 12.2], are explicit code families that can be used in our code constructions in this dissertation.

Many other bounds also consider the field size q in their calculations. Among the best-known bounds are Griesmer (Section 2.4.2), Ball–Blokhuis (Section 2.4.3), Hamming (Section 2.4.4), and Gilbert–Varshamov (Section 2.4.5). Other limits like Bassalygo, Linear Programming, Johnson and Plotkin are not stated in this work, but for the interested reader, we refer to [Bas65], [MS77, Chapter 17] and [Rot06, Chapter 4].

2.4.2 The Griesmer Bound

The Griesmer bound [Gri60] states the length of the shortest linear code over $\mathbb{F}_q$ of dimension k and minimum distance d that increases the right-hand side of the inequality $n \geq d + k - 1$ (the Singleton bound in (2.25)). The Griesmer bound is stated as the following.

Theorem 2.4 (Griesmer Bound [MS77, Thereom 23])**.** *Let $n_q(k > 1, d)$ denote the minimum codeword length required for a linear code over $\mathbb{F}_q$ of dimension $k > 1$ and minimum distance d. Then it holds that*

$$n_q(k > 1, d) \geq d + n_q(k-1, \left\lceil \frac{d}{q} \right\rceil), \tag{2.27}$$

where $n_q(k-1, \lceil d/q \rceil)$ denotes the next possible shortest length for a liner code over $\mathbb{F}_q$ of dimension $k-1$ and minimum distance $\lceil d/q \rceil$.

By (2.27), recursively taking $0, 1, \dots, k-1$ we obtain

$$n_q(k > 1, d) \geq \sum_{i=0}^{k-1} \left\lceil \frac{d}{q^i} \right\rceil. \tag{2.28}$$

As the Griesmer bound relates to the Singleton bound, it is also an *upper* bound on the size and the minimum distance of any linear code. The Griesmer bound is also an improvement on the Hamming bound (described in Section 2.4.4) on the minimal required redundancy, i.e., $n-k$ symbols.

We shall compare in Chapter 6 our formulated bounds to the Griesmer bound for codes consisting of a codeword of weight n and dimension $k > 1$.

2.4.3 The Ball–Blokhuis Bound

Another *lower* bound on the length n of a linear code of dimension k and minimum distance d is the Ball–Blokhuis bound. In fact, it bounds the maximum weight of a codeword as follow.

Theorem 2.5 (Ball–Blokhuis Bound [BB13, Theorem 6.1])**.** *Let $\mathcal{C}$ over $\mathbb{F}_p$ for prime p be a linear code of length n, dimension k, minimum distance d. If $\mathcal{C}$ contains a codeword of weight n, then*

$$n \geq \frac{d}{(p-1)} + d + \theta, \tag{2.29}$$

where $\theta \in \{0, 1, \dots, k-2\}$ is maximal while the following holds:

$$\binom{n-d}{\theta} \equiv 0 \bmod p^{k-1-\theta}.$$

The maximum weight of a codeword $\boldsymbol{c} \in \mathcal{C}$ over $\mathbb{F}_p$ following from (2.29) *is then*

$$\max_{\boldsymbol{c} \in \mathcal{C}} \mathrm{wt}(\boldsymbol{c}) \leq (n-d)p - \theta(p-1).$$

To establish a connection between (2.29) and the Griesmer bound from (2.28) for $q = p^\mu$ with $\mu = 1$, we rewrite $d = \sum_{i=a}^{k-2} d_i p^i < p^{k-1}$, $d_a \neq 0$. Then we subtract the right-hand side of (2.28) from the right-hand side of (2.29) to obtain

$$\frac{d}{(p-1)} + d + \theta - \sum_{i=0}^{k-1} \left\lceil \frac{d}{p^i} \right\rceil = \frac{\left(\sum_{i=a}^{k-2} d_i \right)}{(p-1)} + \theta + 1 + a - k. \tag{2.30}$$

Since θ depends on k and $n-d$ and not directly on d, (2.30) gives either a positive result, which is an improvement on the Griesmer bound, or a negative outcome which is not [BB13, Section 2, page 577]. It is important to emphasize that the Griesmer bound is valid for all linear codes over $\mathbb{F}_q$, whereas the Ball–Blokhuis from (2.29) is only valid for linear codes over a prime field that contain an n-weight codeword, e.g., the all-one word.

We shall use Ball–Blokhuis bound in Chapter 6 to compare our derived bounds for codes having a codeword of weight n.

2.4.4 The Sphere-packing (Hamming) Bound

The sphere-packing bound, or well-known as the Hamming bound [Ham50], packs spheres around each codeword to consider the size of a code. A *sphere* of radius r in $\mathbb{F}_q^n$ is a set of codewords in $\mathbb{F}_q^n$ at Hamming distance r from a fixed codeword in $\mathbb{F}_q^n$. The union of spheres with radii up to r defines a Hamming *ball.* Let $\mathrm{Vol}_q(n,r)$ denote the *volume* of a Hamming ball with radius r, then

$$\mathrm{Vol}_q(n,r) = \sum_{j=0}^{r} \binom{n}{j} (q-1)^j. \tag{2.31}$$

The sphere-packing bound is then expressed in Theorem 2.6, and it is an *upper* bound for linear codes.

Theorem 2.6 (q-ary sphere-packing bound [Rot06, Theorem 4.3])**.** *For any* $[n, k, d]$ *code over* $\mathbb{F}_q$*:*

$$q^k \cdot \mathrm{Vol}_q\left(n, \left\lfloor \frac{d-1}{2} \right\rfloor\right) \leq q^n. \tag{2.32}$$

The radius $r = \lfloor \frac{d-1}{2} \rfloor$ assures *unique decoding* as at most one codeword is at most $\lfloor \frac{d-1}{2} \rfloor$ apart from the received word. Codes attaining the sphere-packing bound in (2.32) with *equality* are named *perfect codes*, e.g., RP and SPC in Section 2.3.5. Perfect codes are well studied and listed in the coding theory books, e.g., [Rot06, page 96].

We shall compare our developed *sphere-packing-type bound* in Chapter 6 to the usual sphere-packing bound given in (2.32), which we name "only errors" in Figure 6.1.

2.4.5 The Gilbert–Varshamov Bound

The Singleton and the sphere-packing limits provide necessary conditions on the parameters of codes. In contrast, the following theorem offers a *sufficient* prerequisite for the existence of a linear code with the provided parameters.

Theorem 2.7 (q-ary Gilbert–Varshamov Bound [Rot06, Theorem 4.4])**.** *Let* $\mathbb{F}_q$ *be a finite field of size* q*, and let* n*,* k*, and* d *be positive integers such that*

$$q^k \cdot \mathrm{Vol}_q(n-1, d-2) < q^n. \tag{2.33}$$

Then the existence of a linear code $[n, k]$ *over* $\mathbb{F}_q$ *with minimum distance at least* d *is guaranteed.*

The q-ary Gilbert–Varshamov bound, in abbreviation q-ary GV bound, is a *lower* bound on linear codes for specific parameters n, k, d, and q.

We employ the q-ary GV bound techniques to show our formulated *GV-like* bounds in Chapter 6 with additional properties. In practical, we also compare to the $(q-1)$-ary GV bound. For short, we occasionally use "GV bound" instead "q-ary GV bound" in the rest of this thesis.

2.4.6 Asymptotic Bounds

All the previous boundaries have their corresponding *asymptotic* versions. Let $[n,k,d]_q$ be a linear code denoted by $\mathcal{C}$ whose size $M = q^k$. The *relative minimum distance*, denoted by δ, of $\mathcal{C}$ is its minimum distance d over its length n such that $\delta = {}^{d}\!/_{n}$ for n heads to infinity, i.e., $n \to \infty$. The relation between δ and the rate $R = {}^{\log_q M}\!/_{n}$ describes the asymptotic behavior of the code $\mathcal{C}$. We direct the interested readers to [Rot06, Chapter 4, Section 4.5] and [MS77, Chapter 17, Section 7] to explore the aforementioned bounds (plus other limits) in their asymptotic expressions. However, we state the asymptotic GV bound in the following section for particular significance since we will use it to derive and compare our *asymptotic GV-like bounds* in Chapter 6.

Asymptotic Version of the q-ary Gilbert–Varshamov Bound

To introduce the asymptotic version of the q-ary Gilbert–Varshamov bound stated in (2.33), we recall the following lemma and its proof that have been stated in many references including [Rot06, page 105] and [GRS19, Proposition 3.3.1] to estimate the volume of a Hamming ball employing the q-ary entropy function.

Lemma 2.2 (Hamming Ball Estimation). *For positive integers n, $q \geq 2$ and real δ, $0 \leq \delta \leq 1 - \frac{1}{q}$,*

$$\mathrm{Vol}_q(n, \delta n) \leq q^{h_q(\delta) n}. \tag{2.34}$$

Proof. It is immediate for $r = 0$. Now for $r > 0$, dividing both sides in (2.34) by $q^{h_q(\delta)n}$ and solving the left-hand side is as follows,

$$\begin{aligned} q^{-h_q(\delta)n} \cdot \mathrm{Vol}_q(n,r) &= \delta^r (1-\delta)^{n-r} (q-1)^{-r} \cdot \sum_{i=0}^{r} \binom{n}{i} (q-1)^i \\ &\overset{\delta \leq 1-\frac{1}{q}}{\leq} \delta^r (1-\delta)^{n-r} (q-1)^{-r} \cdot \sum_{i=0}^{n} \binom{n}{i} (q-1)^i \left(\frac{\delta}{(1-\delta)(q-1)} \right)^{i-r} \\ &= \sum_{i=0}^{n} \binom{n}{i} \delta^i (1-\delta)^{n-i} \\ &= (\delta + (1-\delta))^n = 1, \end{aligned}$$

therefore, $\mathrm{Vol}_q(n, \delta n) \leq q^{h_q(\delta)n}$, which is the claim. □

The asymptotic version of the q-ary Gilbert–Varshamov boundary of linear codes is presented in the next theorem.

Theorem 2.8 (Asymptotic q-ary GV Bound [Rot06, Theorem 4.10])**.** *Let n and nR be positive integers, $\mathbb{F}_q$ be a finite field, and δ be a real in $(0, 1 - {}^1\!/_q]$ that fulfills*

$$R \leq 1 - h_q(\delta).$$

Then a linear code $[n, nR, \geq \delta n]$ over $\mathbb{F}_q$ exists.

Proof. Considering $\mathrm{Vol}_q(n, \lceil \delta n \rceil) \leq q^{n(1-R)}$ in Theorem 2.7 proves the existence of $[n, nR, \geq \delta n]$ code over $\mathbb{F}_q$. Then it follows from Lemma 2.2. □

So far, all the prior bounds are for codes over a single alphabet, i.e., the alphabet q. In the sequential sections, we introduce two-upper bounds versions for *polyalphabetic* codes whose described next.

2.5 Polyalphabetic Codes

Classical linear codes over finite (extension) fields (cf. Section 2.3) are from a *single* alphabet, i.e., alphabet q; thereby, they are discriminated as *monoalphabetic* codes. Any codeword of these codes is of components (coefficients in the polynomial representation) from the field $\mathbb{F}_q$ of size q. However, in realistic applications, the symbols of a codeword could be from multiple alphabet sizes. Applications like orthogonal-frequency-division multiplexing (OFDM) transmission and memory with partially defective positions (described in Section 3.4.3) motivate considering error-correcting codes with different alphabets for every coordinate of a codeword. In these applications, perhaps through a periodic sampling routine, both sender and receiver know which coordinates have smaller alphabet sizes. Sidorenko et al. [Sid$^+$05] define these codes as *polyalphabetic block codes*. Several publications [HS71; EG93; BHOS98; BG04] have already been written about polyalphabetic (or *mixed-alphabetic*) codes. The work in [Sid$^+$05] describes these codes over arbitrary alphabet sizes and do not assume any algebraic structure. Therefore, it provides a generalization compared to other works. We redefine the polyalphabetic codes as follows.

Definition 2.9 (Polyalphabetic Codes)**.** *Let $\mathcal{C}_{pol}$ be a code that consists of a set of codewords of finite length n such that every symbol of a codeword at position i for $i = 0, 1, \ldots, n-1$ belongs its own alphabet q_i. Then*

$$\mathcal{C}_{pol} := \{\boldsymbol{c} \mid \boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1}),\ c_i \in [q_i]\}$$

is a polyalphabetic (or mixed-alphabetic) block code, where $q_0, q_1, \ldots, q_{n-1}$ are alphabets of (possibly) different sizes.

Proposition 2.2 (Properties of Polyalphabetic Codes [Sid+05]). *Let $\mathcal{S}$ be a space that has size $|\mathcal{S}|$, then*

$$|\mathcal{S}| := \prod_{i=0}^{n-1} q_i.$$

$\mathcal{C}_{pol}$ is defined as a subset of $\mathcal{S}$ such that $\mathcal{C}_{pol} \subseteq \mathcal{S}$. The code $\mathcal{C}_{pol}$ has size $|\,\mathcal{C}_{pol}\,|$ and of rate

$$R = \frac{\log |\,\mathcal{C}_{pol}\,|}{\log |\mathcal{S}|}.$$

For any two codewords $\boldsymbol{c}_1, \boldsymbol{c}_2 \in \mathcal{C}_{pol}$, the minimum Hamming distance is given by

$$d = \min_{\substack{\boldsymbol{c}_1, \boldsymbol{c}_2 \in \mathcal{C}_{pol} \\ \boldsymbol{c}_1 \neq \boldsymbol{c}_2}} d_H(\boldsymbol{c}_1, \boldsymbol{c}_2).$$

Definition 2.9 does not restrict the alphabets $q_0, q_1, \ldots, q_{n-1}$ to be prime or power of prime.

2.5.1 Upper bounds on Polyalphabetic Codes

We have defined the mixed-alphabetic codes in Definition 2.9 and showed their proprieties in Proposition 2.2. Next, we exhibit the upper bounds on these codes since they are the cornerstone for our proposed upper bounds in Section 6.2. For more bounds on polyalphabetic codes, we refer to Sidorenko et al. [Sid+05, Theorem 4], Perkins et al. [PSS06], and our *recent results* in [YAPW23] that are briefly summarized in Section 6.4 in Chapter 6.

The Singleton-type Bound on Polyalphabetic Codes

The generalized of the Singleton bound (described in Section 2.4.1) corresponding to mixed-alphabetic codes is given in the theorem below.

Theorem 2.9 (Singleton-type Bound on Polyalphabetic Codes [Sid+05, Theorem 2]). *Let code $\mathcal{C}_{pol}$ be a polyalphabetic code as defined in Definition 2.9 with properties in Proposition 2.2. The size of the code $\mathcal{C}_{pol}$ with distance d is bounded from above by*

$$|\mathcal{C}_{pol}| \leq \prod_{i=1}^{n-d+1} q_i. \tag{2.35}$$

Proof. The proof goes as follows. Assume we obtain a new code, denoted by $\mathcal{C}'_{pol}$, by shortening the code $\mathcal{C}_{pol}$ in at least $d-1$ coordinates. Shortening a code by ℓ locations, i.e., in classical monoalphabetic codes (presented in Section 2.3), reduces the distance of a code by at most ℓ (see, e.g., [MS77, page 29]). A similar argument applies to

$\mathcal{C}'_{pol}$, so its distance becomes at least 1. In other words, code $\mathcal{C}'_{pol}$ codewords of length $n-(d-1)$ should *pairwise* differ; and therefore,

$$|\mathcal{C}_{pol}| = |\mathcal{C}'_{pol}| \leq \Big| [q_1] \times \cdots \times [q_{n-d+1}] \Big| = \prod_{i=1}^{n-d+1} q_i. \qquad \square$$

Theorem 2.9 and its proof will directly contribute in our proof of Corollary 6.1, located in Chapter 6.

The Sphere-packing-type Bound on Polyalphabetic Codes

The authors in [Sid+05, Theorem 3] also developed a sphere-packing-type upper bound based on a simple expression for sphere size with an exponential number of terms.

As for the classical Hamming bound (see Section 2.4.4 and (2.31)) for single-alphabetic codes, the number of words in a *poly-alphabetic* code at Hamming distance r from a given codeword in the center of a *sphere* defines the *volume* of this sphere of a given radius r. Let $V_r^{(s)}$ denote a Hamming *sphere* of radius r. $V_r^{(s)}$ can be obtained by the product of each $(q_i - 1)$ considering q_i for $i = 0, 1, \dots, n-1$ and then summation on all q_i appearing in each radius $0, 1, \dots, r$, so we get

$$\begin{aligned} V_0^{(s)} &= 1 \\ V_r^{(s)} &= \sum_{1 \leq i_1 < \ldots < i_r \leq n} (q_{i_1} - 1) \cdots (q_{i_r} - 1). \end{aligned} \tag{2.36}$$

Note that $V_0^{(s)} = 1$ means a fixed codeword at the center of the sphere is measured with a codeword with $r = 0$ distance apart, i.e., it is measured with itself.

Then, the volume of a Hamming *ball* of radius r, denoted by $V_r^{(b)}$, is a summation of $V_r^{(s)}$ on all possible values $0, 1, \dots, r$ as the following:

$$V_r^{(b)} = \sum_{i=0}^{r} V_i^{(s)}. \tag{2.37}$$

A *sphere-packing-type* bound on polyalphabetic codes is stated in the following theorem.

Theorem 2.10 (Sphere-packing-type Bound on Polyalphabetic Codes [Sid+05, Theorem 3]). *Let code $\mathcal{C}_{pol}$ be a polyalphabetic code as defined in Definition 2.9 with properties in Proposition 2.2. The size of the code $\mathcal{C}_{pol}$ with distance d is bounded from above by*

$$|\mathcal{C}_{pol}| \leq \left\lceil \frac{|\mathcal{S}|}{V_r^{(b)}} \right\rceil, \tag{2.38}$$

where $r = \lfloor \frac{d-1}{2} \rfloor$.

Theorem 2.10 shall be needed to prove Corollary 6.2, provided in Chapter 6.

Our work in this dissertation consider defective memories (Section 3.4.3) that can only utilize partial alphabet sizes at the defective positions, i.e., a subset of the set $[q]$ can be stored on these corrupted memory locations. Thus, upper bounds on size of an arbitrary polyalphabetic code (cf. Section 2.5.1) are used to derive our bounds on codes for partially stuck memory cells in Corollary 6.1 and Corollary 6.2.

In the next chapter (Chapter 3), we proceed to define and explore the reliability problems related to data storage memories. Since these storage units are point-to-point communication systems with discreet memoryless channel (see Figure 3.1), subsequently, Chapter 4 proposes coding methods to overcome reliability issues, e.g., the partially stuck scenario (cf. Section 3.4.3) and substitution errors (e.g., cf. Section 3.3.5).

3 Memories with Defects and Errors

3.1 Introduction

We are in the era of data! The explosion of data has derived the need for storage solutions, ranging from small units to massive storage systems. In turn, it led to revolutionary new data devices, e.g., DNA-based storage [SH22] or other ultra-reliable high-dense units, to compromise the massive growth in data. The dramatic increment in data usage was estimated to reach 40 Zettabytes[1] in 2020 [ZW14]; in reality, it reached 64.2 Zettabytes, with a forecast that by 2025, it will approach 181 Zettabytes [Sta22]. A key feature of storage media, e.g., *non-volatile memories* (NVMs), see Section 3.2, is their longevity as permanent data containers, besides their availability with affordable costs. In contrast, an immediate issue with NVM technologies is their unacceptable reliability due to increased cell levels; simultaneously, the dimensions of the cells were diminished. For instance, contemporary NVMs devices offer increased storage capacity using denser memory components [DS16].

Typically, storage media can be modeled as a communication system, e.g., a point to point communication system with a discreet memoryless channel (DMC) depicted in Figure 3.1. Thus, channel coding principles and error-correcting codes (see Section 2.3 in Chapter 2) can be applied to these memories to ensure *reliable* communication and fault-tolerant information storage and processing.

This chapter is organized as follows. Section 3.2 defines the non-volatile memories and their types, like flash memory and phase-change memories stated in Sections 3.2.1 and 3.2.2, respectively. We see in Section 3.3 why these memories show unreliable behavior, i.e., they encounter various noise and error types listed in Sections 3.3.1, 3.3.2, 3.3.3, 3.3.4, and 3.3.5. Next, Section 3.4 outlines some coding-based solutions to moderate the reliability problems. This chapter, therefore, defines the origin of the reliability problems and motivates us to propose solutions, given in Chapter 4.

[1]One Zettabyte equals $8 \cdot 10^{21}$ bits.

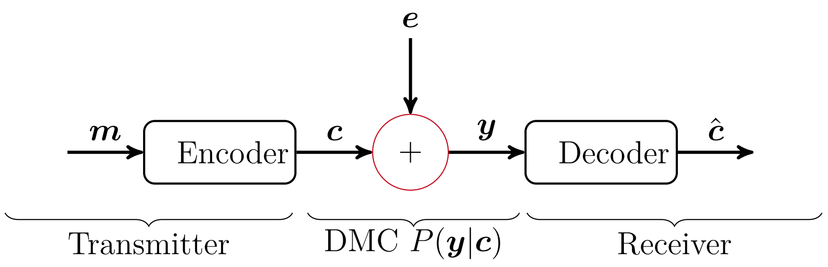


Figure 3.1: Point-to-point communication system with an additive discrete memoryless channel (DMC): a transmitter wants to reliably communicate (store) an information word (or a message) $\boldsymbol{m}$ out of M possible information words at a rate R over a noisy communication channel (or a noisy storage medium). The probability $P(\boldsymbol{y}|\boldsymbol{c})$ is a conditional probability distribution defined for every pair $(\boldsymbol{c}, \boldsymbol{y})$ such that it is the probability the receiver obtaining the estimated version $\hat{\boldsymbol{c}}$ of the transmitted codeword $\boldsymbol{c}$ (i.e., $\hat{\boldsymbol{c}} = \boldsymbol{c}$), given the received noisy version of the codeword $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e}$, where $\boldsymbol{e}$ is the error vector.

3.2 Non–Volatile Memories

Non-volatile memories (NVMs) are a type of storage media that retains stored information even after the power source is removed [DS16]. NVMs have become the principal hard-disk drive substitute for a range of storage applications due to faster data access, lower power consumption, and enhanced physical resilience. The most popular NVM technologies nowadays are flash memories (cf. Section 3.2.1) and phase-change memories (PCM) (cf. Section 3.2.2). Two main procedures are performed to *store* (*write*) data and *retrieve* (*read*) data from these memories. The *writing* process splits further into two sub-tasks: *programming* and *erasing* [DS16]. Programming command means switching a device cell to its higher or lower states except for the erasing state, i.e., the zero value, which contributes the most to the unreliability as it forces a cell to reset to the initial state. These memory units can solely be utilized for their nominal endurance specification, i.e., until specified *program* and *erase* rounds, after which a memory is judged unusable due to unacceptable reliability. In fact, there is a trade-off between the storage density and the endurance, i.e., the lifespan drops ten to twenty times if the cell density doubles [GDS12]. Furthermore, memory reliability rapidly declines due to several physical limitations as storage capacity increases. Applying error-correcting techniques is a particularly interesting strategy for overcoming these physical constraints in NVMs, which is our focus in this thesis.

3.2.1 Flash Memories

The storage unit called *flash memory* has been remarkably developed to increase the storage capacity while keeping the physical dimensions as small as possible. This could be done using the multi-layering concept of flash (also called pages). Flash memory cells are arranged in two-dimensional arrays. Cells are identified by their corresponding bits and are distributed into pages, which are further combined into blocks. Thanks to the previously mentioned flash architecture, it is known that the *programming* mechanism is conducted at the page layer. In opposite, the *erase* procedure is done at the block (containing several pages). Therefore, if the value of a cell needs to be reduced, the block of cells to which it belongs must entirely be wiped and rewritten. As flash cells manufacture from a control gate transistor and a floating gate separated

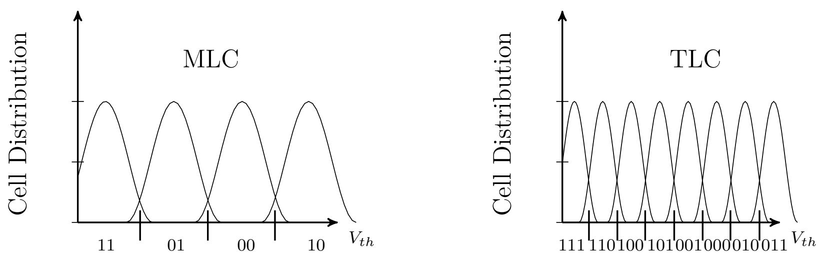


Figure 3.2: Multi-level cell (MLC) and triple-level cell (TLC) with information related to each level. Different cells have various threshold voltage V_{th} requirements, which results in different cells behaviors.

by oxide layers, the amount of charge (voltage level) on the floating gate represents the stored data. Electrons are programmed to and erased from the floating gate through the oxide layer to control the voltage levels (i.e., capturing and releasing phenomena in oxide traps) [WWCW16].

Four measures are commonly used to assess the performance of these storage systems: longevity, latency, throughput, and *reliability.* Longevity is the period of time between the initial storage of data and the point at which it can no longer be recovered. Latency is when a system must finish an action after receiving a command to obtain or store data. Throughput measures how much data the system can typically store and retrieve in a predetermined time frame. *Reliability* is determined by the likelihood that data restored from the system differs from the data initially saved. Reliably storing and retrieving information receive much consideration in these long-term storage devices, which is our focus in this work.

The common types used in the literature to investigate the improvement in the reliability and capacity of these storage units are multi-level cell MLC flash [WWCW16], which contains four possible writing levels, namely $\{0, 1, 2, 3\}$ or their binary representation $\{00, 01, 10, 11\}$, and triple-level cell TLC of eight levels $\{0, 1, \cdots, 7\}$. In principle, each multi-level cell holds one of the q levels and can be considered as a symbol over a discrete alphabet of size q. The lowest level generally refers to the entirely erased state, while the highest corresponds to the maximally programmed phase. Dense flash of sixteen levels is also available [GSD14]. Figure 3.2 illustrates MLC and TLC devices with distribution associated with different cell levels. Apparently, from this figure, the more levels represented in a cell, the lower the device's reliability as the neighboring distributions contour more overlaps.

Since a flash unit is, in practice, a point-to-point communication system with a discrete memoryless channel (see Figure 3.1), the reliability issues could be resolved in the channel codes of flash program/erase (or more generally write/read) channels [WWCW16]. Reducing program/erase (P/E) rounds, specifically delaying or preventing a few erase states, is one direct resolution to which advanced tailored error-correction code schemes could apply. We outline some of these coding-based methods in Sections 3.4.1, 3.4.2, and 3.4.3. Then we provide Chapter 4, which gives our contribution in this direction by proposing coding schemes to prevent the erasing state, i.e., we store codewords with nonzero components in the unreliable cells.

3.2.2 Phase-Change Memories

One other excellent NVM device is a phase-change memory (PCM) technology. Individual cells store digital information via a physical cell state, similar to flash memories. PCM cells have two potential states; an amorphous and a crystalline state; in each, one bit can be stored. The crystalline state can be further programmed to *partially* states, then multiple bits per cell can be stored [Bur$^+$10].

To program a PCM cell to a desired new value, a series of actions referred to as RESET (for erasing) and SET (for programming) must consecutively be conducted. The maximum number of permitted RESET procedures is constrained, just like with flash. The RESET commands a cell to return to the initial amorphous state (the zero state) by applying a significant temperature. The SET operation sets this cell to the desired crystalline value using a more moderate temperature. These cells' cooling and heating processes occasionally cause failures in switching their states. Failure may take place in a position in one of the extreme states or the *partially* programmable states of crystalline in *multi-level* PCM cells. Thus, PCM cells may become *defective* (also called *stuck*) [GPB09; Hwa$^+$05; Che$^+$22; Pir$^+$04], i.e., cells can only hold a single phase [GPB09; Pir$^+$04].

Chapter 4 provides code constructions appropriate to treat these defective cells and any substitution errors that may arise during the programming/erasing or reading mechanisms.

3.3 Channel Model Additive Noise

Unreliability in non-volatile memories (NVMs) evolves thanks to various noise and error types (see Sections 3.3.1, 3.3.2, 3.3.3, 3.3.4, and 3.3.5). According to the amount of charge written into and then removed from the memory cell, reliability deteriorates over time. The term "wear-out" refers to this degradation, which may be considered a time-varying noise whose variance rises with the number of electrons pushed into and out of the floating gate, e.g., in flash devices. It can be dealt with on various levels: at the device level, three-dimensional cell architectures increase durability [CS11; PBK14; Im$^+$15]; or at the system level, channel codes like BCH [CZW08; CKCH14; KH14] and LDPC [MK09; Zha$^+$13; ZPJ10; Wan$^+$14] codes add redundancy to safeguard the stored data.

This thesis focuses on the latter, in which our code constructions in Chapter 4 lie.

3.3.1 Programming Noise

The programming noise denoted by n_p [CWW14; WCW15; TTN96; Com$^+$07] is treated as a Gaussian noise for each level such that the noise variance of the *programmed* states is lower than that of the *erased* state. Thus, assuming x_i is the intended i-th voltage level of a cell and the parameters σ_p and σ_e are the standard deviations from the value x_i, the noise variance of the *programmed* states for $i > 0$ is then given as the PDF (probability density function) by

$$f_{n_p}(n_p|x_i) = \frac{1}{\sigma_p\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x_i}{\sigma_p}\right)^2\right),$$

and for the *erased* state for $i = 0$ is represented as

$$f_{n_p}(n_p|x_i) = \frac{1}{\sigma_e\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x_i}{\sigma_e}\right)^2\right),$$

where $\sigma_e > \sigma_p$. For MLC and TLC storage media, the *zero* level represents the erased state [WWCW16]. The zero subscripts must be avoided in the output vector before storing it in the memory to prevent the need to erase the current data and avoid the most prominent contribution of the programming noise (as $\sigma_e > \sigma_p$). Hence, and as later emphasized in Section 3.4.3, the case of partially stuck at-1 (equivalently saying 0 is not allowed) is of a special consideration.

3.3.2 Wear-Out Noise

In NVMs, capturing and releasing charges during programming and erasing leads to *wear-out* in the form of charge trapping in the oxide and interface states [CWW14;

WCW15; ORS86; MC+09], which is a form of permanent damage. The trap (the *stuck-at*; see Section 3.4.3) prevents a cell from changing its value, despite new charges being injected or removed from this cell. This noise figure can be modeled as a positive-side exponential [WWCW16, Section A.3], a negative-side exponential, or a double-sided exponential (Laplace) distribution [CWW14; PPMP14]. Letting n_w be the wear-out noise. The PDF (for the positive-side exponential) for each level is then expressed as

$$f_{n_w}(n_w) = \begin{cases} \frac{1}{\lambda}\exp(-\frac{n_w}{\lambda}) & \text{if } n_w \geq 0 \\ 0 & \text{if } n_w < 0 \end{cases},$$

where λ denotes the *wear-out decay constant* that characterizes the *slope* of the distribution (cf. Figure 3.2 showing the standard distribution), and λ grows proportionally with the number of program/erase rounds.

3.3.3 Retention Noise

Retention noise denoted by n_r is another type of noise disturbance developing reliability issues that result from trapping recovery and charge (electron) detrapping [Mie+06; Yan+06; LCPK03; CWW14; WCW15; Mie+04; Don+12]. It models the threshold voltage integrity degradation due to *charge leakage* after a charge is written. It can approximately be expressed as Gaussian random variable with PDF as follows.

$$f_{n_r}(n_r|x) = \frac{1}{\sigma_r\sqrt{2\pi}}\exp\left(-\frac{1}{2}\left(\frac{x-\mu_r}{\sigma_r}\right)^2\right),$$

where μ_r denotes the *mean of the distribution* and σ_r denotes the *nominal deviation* from the value x, where x is the cell voltage affected by the charge leakage. Both μ_r and σ_r depend on the required threshold voltage, retention time, and the number of (P/E) cycles.

3.3.4 Cell-to-cell Interference

Usually, any memory cell suffers writing or reading errors. One of the errors related to the writing process is *inter-cell interference* [Don+12; DLZ10; SC19a]. Writing on memory cells requires increasing or decreasing their corresponding voltage values to write the desired q-ary levels. This triggers a phenomenon, called inter-cell interference (ICI), that causes magnitude-1 (*Mag-1*) or its *asymmetric* version *Amag-1* errors. In non-volatile memories like flash, Mag-1 defines transitions of "one level" *upward* or *downward* from the correct symbol, and Amag-1 describes "one level" changeovers in the *upward* direction *only* [SC19a]. For any cell-i, this process could alter the adjacent cells, e.g., cell-$i-1$ and cell-$i+1$ by inducing their instantaneous voltage levels such that they might have higher or lower magnitudes, as shown in Figure 3.3.

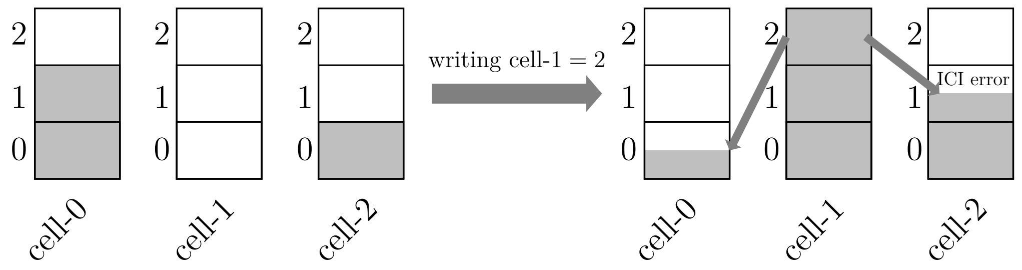


Figure 3.3: Illustration of inter-cell interference in non-volatile memories. Writing on cell-1 induces the neighboring cells by altering their prior voltage levels. Due to writing the value 2 to cell-1, cell-0 shows ICI (Mag-1) error as its level decreased from 1 to 0. As a similar consequence, cell-2 suffers Mag-1 or Amag-1 corruption as its value incremented from 0 to 1.

3.3.5 Programming Errors

Programming error are the dominant cause of unreliability in scaled NVMs devices when they are cycled excessively [PPMP14; Hel+14]. It is the error in which a cell appears to have been programmed to a different state than the intended one. Such events may result from an erase failure or the two-step programming process frequently employed in MLC flash storage.

Assuming no additional noise components and that the original data is distributed uniformly at random (i.e., each level is equally *likely*), programming errors can raise some *write* thresholds voltages to higher levels, making them equally *unlikely*. Thus, it alters the distribution of the stored data. To model the programming error, the probability mass function PMF can be used. For example, for each level $l_x \in \{0, 1, 2, 3\}$ in MLC devices, the conditional probability of actually writing y for level l_y is

$$\sum_{l_y=0}^{3} P_{l_x,l_y} = 1,$$

where $P_{l_x,l_y} = P(L_y = l_y | L_x = l_x)$.

Endurance enhancement may be achieved in these memories if such programming errors have been prevented, e.g., by error-correcting (cf. Section 2.3) and defects-masking (cf. Section 3.4.3) codes, as proposed in Chapter 4.

3.4 Coding Methods for Non–Volatile Memories

Before we command a new *write*, a *read* process needs to be applied earlier to determine the instantaneous cell status, then decide upon the new data while writing. For example, let a cell have two bits (LSB and MSB[2]) with values "1" and "0", respectively (e.g., both bits represent the integer value $2 \in [4]$ considering MLC cell). Programming them to new values "1" and "1" (equivalently writing the integer value $3 \in [4]$) in the recent write requires knowing that the MSB currently has "0" and the programming command only needs to set it to "1". The LSB, on the other hand, does not change. The programming process could then successfully present a new data *without* committing an erasing task; thereby, solutions like in Section 3.4.1, Section 3.4.2, and Section 3.4.3 are emerged. In these sections we outline some coding-based methods to cope with the reliability concerns, i.e., reducing the number of program and erase (P/E) cycles by avoiding unnecessary *erase* states (equivalently, returning back to the *zero* value is not needed).

3.4.1 Write Once Memory

Write once memory (WOM) codes [RS82; Jia07; Yaa$^+$12; GD15; SC19b] provide one approach to reduce the number of P/E cycles required to store information by permitting multiple writes before an erase cycle is needed. As a consequence, it delays the deterioration and results in higher device reliability. Solomon et al. in [SC19b] combine WOM codes and error-correction codes by concatenating codes with usual disturbance due to noise (see Sections 3.3.1, 3.3.2 and 3.3.3) or inter-cells interference (cf. Section 3.3.4) that causes substitution errors. WOM concept (as a part of a channel) converts many errors to erasures as known correcting erasures (check Section 2.3.2) is easier and more tempting than correcting errors (explained in Section 2.3.3).

3.4.2 Rank Modulation

Another way to defer or protect against the wear-out process in NVMs is rank modulation [JMSB08; JSB08; MBZ13; QJS13]. Rank modulation uses the *relative* value (or *ordering*) of the cell charge levels instead of the *absolute* (in contrast; see Section 3.4.3 and Remark A.3) value to retain information. The *rank* of a cell indicates the relative position of its own charge level such that the ranks of n cells influence an ordered set of $\{1, 2, \ldots, n\}$. Thus, the programming process starts with the lowest-ranked cells, followed by the next lowest-ranked ones, until the highest-ranked cells. As a result, adding charge to reorder the cells correctly makes it possible to *rewrite* a block of cells without *erasing* procedure. Jiang et al. [JSB08] studied the properties of error correction in rank modulation codes to eliminate the need for discrete cell levels and to overcome *overshoot*[3] errors and *asymmetric* errors (e.g., Amag-1 error; see Section 3.3.4).

[2]MLC devices have two bits only in each cell to represent the data, namely the least significant bit (LSB) and the most significant bit (MSB) in the lower and upper *pages*, respectively. On the other hand, TLC memories use three bits per cell, LSB, CSB, and MSB, where CSB stands for the center significant bit [DS16, Chapter 3].

The authors in [JMSB08] also employ the rank modulation and take a single cell of the ordered set of n cells and make it the *top-charged* cell (i.e., choosing an upper charging limit) to avoid charge over-injection[3] during programming processes. Similarly, a *block deflation*[4] might be applied. Thus, the relative cells' values will be preserved as they are scaled by the same constant, adding or subtracting. As a result, the designated *erase* step could be eliminated.

3.4.3 Masking Cells

In NVMs like PCMs, as mentioned in Section 3.2.2, cells might fail to switch their states due to heating and cooling operations while erasing or programming them, thereby becoming *defective*. If they are *completely unable* to change their values, i.e., among q possible values (neither upward nor downward), then these cells are *fully* stuck (see Definition 3.1); in short, we refer to them as *stuck memory cells* (SMC). Otherwise, cells might be unable to utilize one or more states and can only program *partial* levels. Thus, they are termed *partially stuck memory cells* (PSMC) (see also Definition 3.2). We interchangeably use the words "defective" and "stuck" cells in this dissertation.

Corresponding to the two steps, *write* (of sub-tasks *program* and *erase*) and *read*, for example, in MLC (see Section 3.2.1), the *erase* state is considered as the *zero* level, among the four possible writing levels. Note that a partially defective cell at level 0 is a *non-defective* cell that can store any of the q levels, and a partially defective at level $q-1$ is a *fully* defective cell. Hence, avoiding the zero value in the output vector from an encoder of PSMC means preventing the need for a new erase command. Therefore, partially stuck at 1, equivalent to disregarding the 0 value in the output vector, is of special importance and regarded separately in Chapter 4.

Works like [Hee83] and [WY16], employ *masking* techniques using error-correcting codes to overcome these erroneous cells. "Masking" means selecting a word whose entries correspond to writable levels in the (partially) stuck positions. That said, the authors in [WY16], for example, ceased to be able to correct any other error types (apart from the partially stuck-at error) that might arise from the prior mentioned noise figures and error types. On the other hand, Heegard in [Hee83] proposed coding schemes with considerably larger check symbols (i.e., at least the number of stuck-at cells) to overwhelm the defects and other appearing errors, utilizing *partitioned cyclic codes* as discussed in Definition 2.6.

[3]An overshooting error occurs when programming cells, i.e., injecting undesired charges in a cell over its predefined doable programmable levels (correspondingly, over its predefined voltage thresholds).

[4]A block deflation is a process in which there is a possibility of decreasing all the levels in a block of cells by a *constant* amount of charge no exceeding the *lowest* allowed prescribed charge level (i.e., a reverse of taking a top-charged cell) [JSB08].

Defective (Stuck) Memory Cells SMC

Traditional coding for memory with defects originated in [KT74] that later initiated a series of publications [TGKO75; Tsy75b; Tsy75a; Tsy77; BS77; LKD78; KKY78; Hee83; Kuz85; Che85; BV87; Dum87; Dum89; Dum90] under different facets. We define the *fully stuck memory cells* as the following.

Definition 3.1 (Stuck Memory Cells SMC)**.** *A cell is called defective (*stuck-at level *s), if it can only store the value s.*

Partially Defective (Stuck) Memory Cells PSMC

The more flexible and relatively common case is the *partially stuck memory cells* (refer also to the beginning of Section 3.4.3). We explicitly state the following definition for *partially stuck memory cells.*

Definition 3.2 (Partially Stuck Memory Cells PSMC)**.** *A cell is called partially defect (*partially-stuck-at level *s), if it can only store values which are at least s.*

Figure 3.4 depicts the general idea of reliable and (partially) defective memory cells. It illustrates two different cell level representations: Representation 1 forms the binary extension field $\mathbb{F}_{2^4}$ and Representation 2 forms the set of integers modulo $q = 4$, e.g., $\mathbb{Z}/4\mathbb{Z}$ (see Section 2.1).

Notice that it is not in our concern which (exact) one of the former noise/error types (cf. Section 3.3.1, Section 3.3.2, Section 3.3.3, Section 3.3.4, and Section 3.3.5) causes the partially stuck situation in a cell or imposes substitution errors. However, we have explained how these noise and error figures result in unreliability and pose errors on NVMs, to which Chapter 4 provides coding solutions.

Our suggested coding methods (see Chapter 4) that utilize the partial levels of the partially defective cells have the benefit of requiring fewer parity symbols than *naively* carrying the information about the positions of the faulty cells to the decoder, thereby providing an advantage over conventional coding approaches, e.g., correcting them as erasures (cf. Section 2.3.2 and Section 4.3).

3.4.4 Unreachable Memory Cells UMC

Unreachable memory cell restriction is seen as a *dual* problem for partially stuck memory cells PSMC (defined in Section 3.4.3) [GSD14; WY16]. Regarding PSMC, Figure 3.4 shows that cells cannot use levels from *below* as they are partially stuck above them. In the previous regard, a cell is said to be *unreachable* at a level (from *above*) wherever it is allowed to store at most that level. For example, a cell is unreachable at the $q-2$ level (assuming q programmable alphabets), thereby permissibly programming it until reaching its $q-2$ value, see Definition 3.3. Figure 3.5 presents an MLC memory (defined in Section 3.2.1) with two unreachable cells.

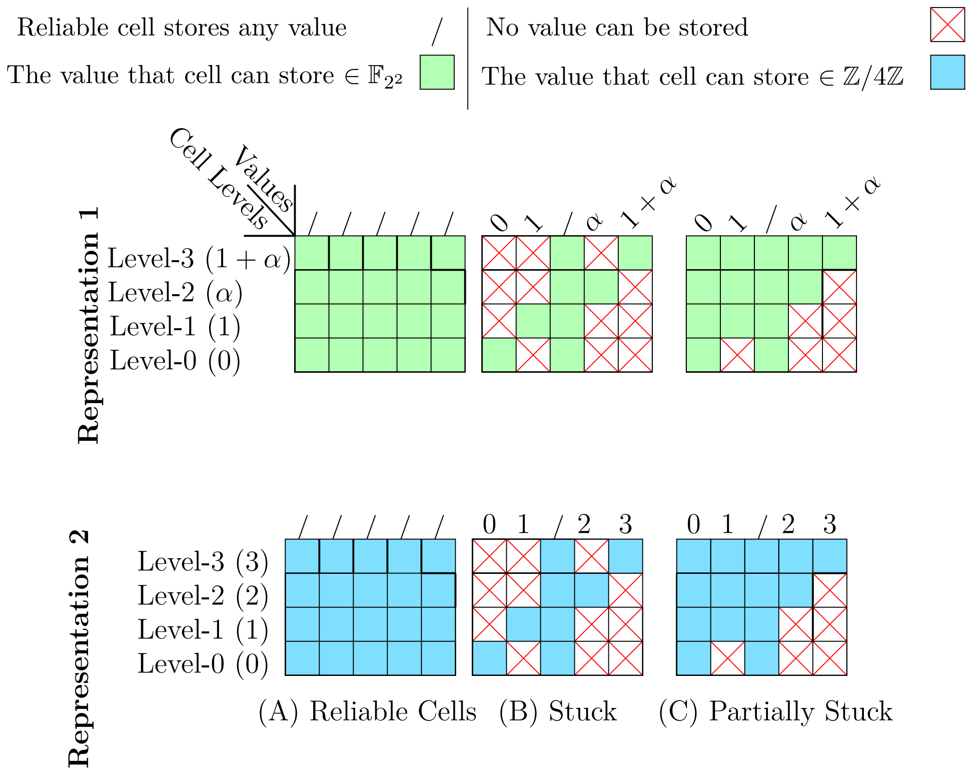


Figure 3.4: Illustration of reliable and (partially) defective memory cells. In this figure, there are $n = 5$ cells with $q = 4$ possible levels. The cell levels $\in \mathbb{F}_{2^2}$ are mapped to (0, 1, α or $1+\alpha$) shown in Representation 1 or $\in \mathbb{Z}/4\mathbb{Z}$ are mapped to $(0, 1, 2$ or $3)$ shown in Representation 2. Case (A) illustrates fully reliable cells which can store any of the four values in both representations. In the stuck scenario as shown in case (B), the defective cells can store only the exact stuck level s. Case (C) is more flexible (partially defective scenario). Partially stuck cells at level $s \geq 1$ can store level s or higher.

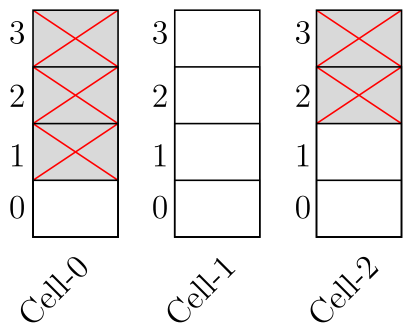


Figure 3.5: A multi-level cell (MLC) device (defined in Section 3.2.1) with two unreachable cells where only their *lower* levels are writable. Cell-0 is reachable at level 0 only, and Cell-2 is programmable at levels 0 and 1. In contrary, Cell-1 is a normal cell and reachable at all its four possible values.

Definition 3.3. *(Unreachable Memory Cells UMC) A cell is said unreachable at level $\tilde{s}$, if it can only store values which are at most $\tilde{s}$ value.*

Reversely to PSMC, an unreachable cell at level $q-1$ is a *normal* cell that can be programmed until the $(q-1)$-th level, and an unreachable cell at level 0 is a *fully unreachable* cell, i.e., it can *only* store the 0 value (e.g., Cell-0 depicted in Fig. 3.5). We intentionally notate with $\tilde{s}$ in Definition 3.3 to show the duality corresponding to s (stated for PSMC in Definition 3.2) that serves our proposed code construction in Section 4.7.1 of Chapter 4, covering this type of memory.

4 Coding Schemes for Memories with Defects and Errors

Abstract

We present new constructions that are able to mask u partially stuck cells while correcting at the same time t substitution errors. The purpose of "masking" is determining a word whose entries coincide with writable levels at the (partially) stuck cells. For $u > 1$ and alphabet size $q > 2$, our new constructions improve upon the required redundancy of known constructions for $t = 0$, and require less redundancy for masking partially stuck cells than former works required for masking fully stuck cells (which cannot store any information).

Some work in this chapter is based on our works in [APW19] that has been published in the 2019 XVI International Symposium "Problems of Redundancy in Information and Control Systems" (REDUNDANCY), in [APW20] that has been published in the 2020 Algebraic and Combinatorial Coding Theory (ACCT), in [APW21] that has been accepted in the 12th Annual Non-Volatile Memories Workshop (NVMW 2021), in [APTW23b] that has been accepted in the 14th Annual Non-Volatile Memories Workshop (NVMW 2023), and in [APTW23a] that is in revision for publication in the journal Designs, Codes and Cryptography (DCC), 2023.

4.1 Introduction

In Chapter 3, we explored main types of noise and reasons contributing to storage media's unreliability. This chapter is to propose new code constructions considering *unreliable* (*partially defective*) memories that also suffer from random errors.

Classical coding for memories with defects (or stuck cells, see Definition 3.1), also known as *defect-correcting codes* for *memories with defects*, dates back to the 1970s, initiated by Kuznetsov and Tsybakov [KT74]. They proposed binary defect-correcting codes in finite and asymptotic regimes whose required redundancy is at least the number of defects. Later works [TGKO75; Tsy75b; Tsy75a; BS77; LKD78; KKY78; Hee83; Kuz85; Che85; BV87; Dum87; Dum89; Dum90] investigated the problem of defective cells under various aspects: binary and non-binary, only defect-correcting coding and error-and-defect-correcting coding, and finite and asymptotic length analysis.

In binary defect-correcting coding models, e.g. [Tsy75b; LKD78; KKY78; Che85; BS77; BV87; Dum87], the authors dealt with masking stuck cells without considering additional substitution errors. In these studies, it is unclear if the proposed constructions are optimal in terms of their required redundancy. The works [Tsy75a; Hee83; Kuz85] considered masking stuck memory cells while at the same time correcting potential random errors. In [Hee83], so-called partitioned cyclic code and partitioned BCH codes were proposed for this task.

The asymptotic model of stuck-cell-masking codes also received considerable attention in the previously mentioned papers. Moreover, there is work devoted to asymptotically optimal codes for a fixed number of defects [Dum89] or for a number of defects proportional to the codeword length [Dum90]. The proposed constructions, for example [Dum90, Section 4] and its extended version in [Dum90, Section 5] that can additionally correct substitution errors, show that u check symbols are sufficient for masking u defects. However, they use codes with a property that is not well studied in coding theory. Therefore, we do not dwell on [Dum89] and [Dum90], and also our goal is to obtain code constructions for finite code length n.

The recent work [WY16] considers *partially* stuck memory cells (cf. Section 3.4.3 and Figure 3.4.C), and improves upon the redundancy necessary for masking compared to all prior works for conventional stuck cells (see Section 3.4.3). However, this paper does not consider error correction in addition to masking. Thus, the authors in [WY16] capture the influence of *one* of the reliability issues, e.g., the wear-out noise (Section 3.3.2). That being said, the encoding algorithms that successfully produce vectors matching the partial defects might fail to appropriately write these vectors to memories with defective positions, or the reading process might be unsuccessful [SC19a]. This happens due to substitution errors caused by inter-cell interference noise (Section 3.3.4) or other noise/error disturbance (Sections 3.3.1, 3.3.2, 3.3.3, 3.3.5). For instance, the read voltage threshold for each cell's level could be distorted, which misleads the writing process (i.e., in the new write), and raises the programming (writing) error introduced in Section 3.3.5, a dominant error in flash devices. Hence, code constructions that overcome the effect of partial defects in the cells and any other channel-added errors are desirable and are proposed in this chapter.

4.1.1 Contributions and Outline

We regard the problem of combined error correction and masking of *partially stuck memory cells*. Compared to the conventional stuck-cell case in [Hee83] as defined in Section 3.4.3, we reduce the redundancy necessary for masking, similar to the results in [WY16], and even reduce further compared to [WY16, Construction 5].

If cells are partially stuck at level 1, we can simply use a $(q-1)$-ary error correcting code as mentioned in [WY16, Section III]. However, this approach could require too much redundancy if a cell is partially stuck at different levels rather than 1. For instance, using $(q-s)$-ary codes for $2 \leq s \leq q-1$ reduces the cardinality of the code because exempting s out of the available q levels is quite expensive. Further, for relatively few partially stuck-at-1 cells, even a $(q-1)$-ary error correcting code is not a competitor to our constructions (cf. Figure 6.5). Therefore, considering sophisticated coding schemes is favorable.

The structure of this chapter is as follows. Section 4.2 recalls a regular erasure pattern's definition to discriminate from the (partially) stuck-at pattern. Section 4.4 then reformulates the classical defective cells theorem (restated in [WY16, Theorem 1] and initially provided by [KT74]) for completeness, as our encoding and decoding schemes in this work employ similar techniques.

Cells are partially stuck at level 1 means the zero values are prohibited (see Figure 3.4). This situation requires no need to commit to the *erasing* state (an undesirable and costly process; see Sections 3.2.1 and 3.2.2) while writing on these cells, i.e., as explained in Section 3.4. Furthermore, it is observable that the *erasing* process contributes the most to the reliability concerns [PPMP14].

Therefore, the first part of this study describes (Definition 4.2) and addresses (Sections 4.5.1 and 4.5.2) the case of being partially stuck at level 1 independently. Section 4.5.1 supplies our construction that remedies substitution errors while masking ("masking" is defined in Section 3.4.3) *at most* $q-1$ partially stuck at-1 cells out of all system cells—followed by a variant version of this construction explicitly using *partitioned* cyclic codes (cf. Section 2.3.4 and Definition 2.6). Section 4.5.2 offers a *probabilistic masking* method that allows utilizing the memory cells with a particular probability even if there are more than $q-1$ partially stuck cells. It further presents two other constructions (with few extensions, examples and remarks) to mask *more than* $q-1$ partially stuck at-1 cells and correct errors. Next, Section 4.6 generalizes our error correction and masking code schemes to any *arbitrary partially stuck-at levels.*

The penultimate part of this chapter, Section 4.7.1, is dedicated to *equivalent* codes for *unreachable memory cells* (defined in Section 3.4.4) to complete our work regarding coding for memories with partially defective levels. The last part, Section 4.8, remarks on the error positions in the case that they overlap with the partially stuck locations but *do not coincide* with the partially stuck restriction.

Overall, this thesis provides code constructions for *any number of partially stuck cells at any stuck-at levels while correcting substitution errors in the Hamming metric*; see Table 4.3 for an overview of our constructions and their required redundancies. For the error-free case, where only masking is necessary, our redundancies coincide with those from [WY16] or are even smaller.

4.2 Regular Erasure Patterns

Generally, the correctability of an erasure pattern (see Section 2.3.2) must fulfill a necessary condition: the number of parity symbols must be at least the number of erasures left in the code truncated in a particular set of locations. The term *regularity* is used to show the sufficiency of the aforementioned condition that was initially established by [GHSY12]. In fact, further conditions on the erasure pattern must be fulfilled such that the pattern is *always* correctable. For larger topologies like flash cells (cf. Section 3.2.1), Proposition 2 in [HPYW21] verifies that not only the prior condition must be satisfied, but also an erasure pattern $\mathbb{E} \subset [n]$ is *correctable-guranteed* if and only if it happens in the parity symbols of a codeword.

4.3 (Partially) Stuck at Patterns

The similarity between regular erasures in the previous section and fully stuck-at patterns (cf. Section 3.4.3) is that storing information is not feasible in both. Regular erasure patterns happen in the channel and are handled in the *decoder* with a condition that they are *necessarily occurring in the check symbols*, as mentioned earlier in Section 4.2. On the contrary, the conventional stuck-at-cell scenario could be resolved in the *encoder* by masking the defective places *anywhere* in a given memory using restricted coordinates of an encoded codeword, given in the following section.

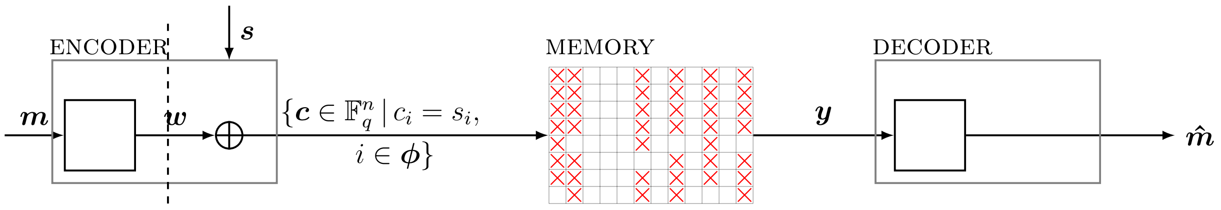
The encoder could avoid the stuck-at positions while writing onto the memory and let the decoder correct them as an erasure pattern (cf. Section 2.3.2), given that the decoder is capable of identifying the stuck-at cells as an erasure pattern, e.g., by a read process. The latter alternative, however, costs considerable redundancy and demands the stuck-at pattern occurring in a *particular set* of locations (i.e., they should not happen in the *information set* of a code as demonstrated in Section 2.3.2) to be correctable [HPYW21, Proposition 2].

Furthermore, the cost of treating *partially* stuck cells like erasures is highly unnecessary as the memory can still *store partial information* in the partially stuck-at positions, which is our interest in this work. For example, codes with a *single redundancy symbol* can be used [WY16].

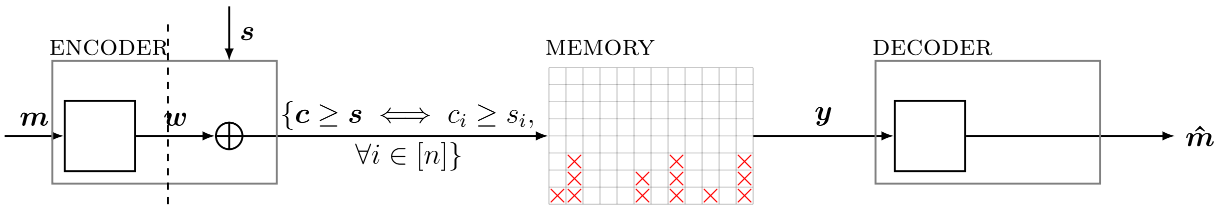
Hence, our methods in this chapter rely on the encoder knowing the (partially) stuck-at pattern to fix them. In contrast, the decoder does not know or care about them. The decoder's assignment is to reassemble the message using only the retrieved codeword (see the decoding principles in Section 2.3.3). Due to the linearity, more than one codeword can represent the same message to recover from (partially) stuck-at errors. That is, for any position indexed as partially stuck-at, multiple possible values (equal or higher than) the corresponding stored values can be applied. Figure 4.1 describes general encoding and decoding schemes for memory with defects.

Legend

$\boldsymbol{m}$: message vector
$\boldsymbol{w}$: augmented message vector
$\boldsymbol{\phi}$: stuck positions, $i \in \boldsymbol{\phi} \subseteq [n]$
$\boldsymbol{s}$: (partially) stuck at levels vector
$\boldsymbol{c}$: encoder output vector sent (transmitted) to the memory by the *write* process
$\boldsymbol{y}$: decoder input vector obtained (received) from the memory by the *read* process
$\hat{\boldsymbol{m}}$ retrieved (restored) message vector



(a) Stuck cell case for a memory consists of n cells.



(b) Partially stuck cell case for a memory consists of n cells.

Figure 4.1: General coding model for memory with defects (cf. Chapter 3). Section 2.1 and Section 2.3 define the legend. Case 4.1a and Case 4.1b illustrate the stuck memory cells and partially stuck memory cells encoding and decoding procedures, respectively. The red cross "×" represents the forbidden levels in the memory.

4.4 Construction for Stuck Cell (Without Errors)

Typically, codes treating the fully stuck cells (see Section 3.4.3 and Definition 3.1) can be defined as follows.

Definition 4.1 (Codes for Stuck Memory Cells $(n, M)_q(\boldsymbol{\phi}, \boldsymbol{s})$ SMC)**.** *For $\boldsymbol{s}^{(\boldsymbol{\phi})} \in \Sigma \subset \mathbb{F}_q^n$ and the set of stuck locations $\boldsymbol{\phi} \subseteq [n]$, a q-ary $(\boldsymbol{\phi}, \boldsymbol{s})$*-stuck-at-masking code *$\mathcal{C}$ of length n and size M is a coding scheme consisting of a message set $\mathcal{M}$ of size M, an encoder $\mathcal{E}$ and a decoder $\mathcal{D}$ satisfying:*

1. *The encoder $\mathcal{E}$ is a mapping from $\mathcal{M} \times \Sigma$ to $\mathbb{F}_q^n$ such that*
$$\text{for each } (\boldsymbol{m}, \boldsymbol{s}^{(\boldsymbol{\phi})}) \in \mathcal{M} \times \Sigma, \quad \mathcal{E}(\boldsymbol{m}, \boldsymbol{s}^{(\boldsymbol{\phi})}) = \{\boldsymbol{c} \in \mathbb{F}_q^n \,|\, c_i = s_i, i \in \boldsymbol{\phi}\},$$
2. *It holds that*
$$\mathcal{D}(\mathcal{E}(\boldsymbol{m}, \boldsymbol{s}^{(\boldsymbol{\phi})})) = \boldsymbol{m}.$$

Construction 4.1. *[WY16, Construction 1] Given $\boldsymbol{s}^{(\boldsymbol{\phi})} \in \Sigma \subset \mathbb{F}_q^n$, where the set $\boldsymbol{\phi} \subseteq [n]$ defines the locations of defective cells. Assume that there is an $[n, k, |\boldsymbol{\phi}| + 1]_q$ code $\mathcal{C}$ with a systematic $(n-k) \times n$ parity-check matrix*

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{I}_{n-k} & \boldsymbol{A}_{(n-k)\times k} \end{bmatrix},$$

where $\boldsymbol{I}_{n-k}$ is the $(n-k) \times (n-k)$ identity matrix, and $\boldsymbol{A} \in \mathbb{F}_q^{(n-k)\times k}$. Encoder and decoder are shown in Algorithm 1 and Algorithm 2.

Theorem 4.1 (Masking Only)**.** *The coding scheme in Construction 4.1 is a $(\boldsymbol{\phi}, \boldsymbol{s})$ SMC of length n and cardinality q^k which needs $n-k$ redundancy.*

Proof. Let $|\ \boldsymbol{\phi}\ | = u$, the proof is then identical to the proof of [WY16, Theorem 1] considering one-to-one mapping $\mathbf{F} : [q] \to \mathbb{F}_q$ so operations like addition and multiplication on $x \in [q]$ are defined as $\mathbf{F}(x)$. □

It is unknown whether Construction 4.1 is ideal in terms of the realized redundancy for masking. However, because there are $|\ \boldsymbol{\phi}\ |$ *fully* defective cells that cannot store information, the redundancy $n-k$ of such a code must be at least the size of $\boldsymbol{\phi}$.

Example 4.1. *[WY16, Example 5] confirms that to mask $|\ \boldsymbol{\phi}\ | = q = 5$, one can use $[30, 22, 6]_5$ code as the best known code with minimum distance $d = |\boldsymbol{\phi}| + 1$, which needs $n - k = 30 - 22 = 8$ check symbols.* ▶

The major drawback in SMC schemes is that they necessitate parity symbols at least the number of stuck-at positions, which is relatively large, e.g., Example 4.1 manifests to mask only five stuck-at cells, a code with the redundancy of eight symbols is used.

We move now to *partially* stuck cell constructions that require *fewer* check symbols to mask these cells, our focus in this dissertation.

```
Algorithm 1: Encoding ($\boldsymbol{m}; \boldsymbol{s}^{(\phi)}$)
  Input:
      • Message: $\boldsymbol{m} = (m_0, m_1, \ldots, m_{k-1}) \in \mathbb{F}_q^k$
      • The vector: $\boldsymbol{s}^{(\phi)} \in \mathbb{F}_q^n$
1 Augment $\{\boldsymbol{m} \mid \boldsymbol{w} = (\boldsymbol{0}_{n-k}, \boldsymbol{m}) \in \mathbb{F}_q^n\}$
2 Find $\{\boldsymbol{z} \in \mathbb{F}_q^{n-k} \mid \boldsymbol{y} = \boldsymbol{w} + \boldsymbol{z}\boldsymbol{H}, w_i = s_i \text{ for } i \in \boldsymbol{\phi}\}$
  Output: Codeword $\boldsymbol{y} \in \mathbb{F}_q^n$ with $y_i = s_i$ for $i \in \boldsymbol{\phi}$
```

```
Algorithm 2: Decoding $\boldsymbol{y}$
  Input:
      • Retrieve $\boldsymbol{y} \in \mathbb{F}_q^n$
1 $\hat{\boldsymbol{z}} \leftarrow (y_0, y_1, \ldots, y_{n-k-1})$
2 $\hat{\boldsymbol{w}} = (\hat{w}_0, \hat{w}_1, \cdots, \hat{w}_{n-1}) \leftarrow (\hat{\boldsymbol{y}} - \hat{\boldsymbol{z}} \cdot \boldsymbol{H})$
3 $\hat{\boldsymbol{m}} \leftarrow (\hat{w}_{n-k}, \ldots, \hat{w}_{n-1})$
  Output: Message vector $\hat{\boldsymbol{m}} \in \mathbb{F}_q^k$
```

4.5 Constructions for Partially Stuck Cell (With Errors)

Non-volatile memories are made of a huge number of cells, and in each one, a few programmable levels are allowed. Therefore, they require channel codes with extremely long block lengths over a relatively small number of alphabets sizes, e.g., multi-level cells (MLC) (cf. Section 3.2.1) employ the alphabet $q = 2^2$. Hence, our code constructions are over small alphabets, i.e., the code length n is larger than the field size q. Otherwise, by using, e.g., [Sol74], one could instead mask by a code of length $n < q$.

We next define codes for the partially stuck cell (see Section 3.4.3 and Definition 3.2).

Definition 4.2 (Codes for Partially Stuck Memory Cells $(n, M)_q(\Sigma,\, t)$ PSMC)**.** *For $\Sigma \subset \mathbb{F}_q^n$ and non-negative integer t, a q-ary $(\Sigma,\, t)$-*partially-stuck-at-masking code $\mathcal{C}$ *of length n and size M is a coding scheme consisting of a message set $\mathcal{M}$ of size M, an encoder $\mathcal{E}$ and a decoder $\mathcal{D}$ satisfying:*

1. *The encoder $\mathcal{E}$ is a mapping from $\mathcal{M} \times \Sigma$ to $\mathbb{F}_q^n$ such that*

$$\textit{for each } (\boldsymbol{m}, \boldsymbol{s}) \in \mathcal{M} \times \Sigma, \quad \mathcal{E}(\boldsymbol{m}, \boldsymbol{s}) \geq \boldsymbol{s},$$

2. *For each* $(\boldsymbol{m}, \boldsymbol{s}) \in \mathcal{M} \times \Sigma$ *and each* $\boldsymbol{e} \in \mathbb{F}_q^n$ *such that*

$$\mathrm{wt}(\boldsymbol{e}) \leq t \text{ and } \mathcal{E}(\boldsymbol{m}, \boldsymbol{s}) + \boldsymbol{e} \geq \boldsymbol{s},$$

3. *It holds that*

$$\mathcal{D}(\mathcal{E}(\boldsymbol{m}, \boldsymbol{s}) + \boldsymbol{e}) = \boldsymbol{m}.$$

Definition 4.3 (Special Case of Definition 4.2: $(n, M)_q(u, 1, t)$ PSMC)**.** *A* q*-ary* $(u, 1, t)$ *PSMC of length* n *and cardinality* M *is a* q*-ary* (Σ, t) *PSMC of length* n *and size* M *where*

$$\Sigma = \left\{ \boldsymbol{s} \in \ \{0,1\}^n \,\middle|\, \mathrm{wt}(\boldsymbol{s}) = |\operatorname{supp}(\boldsymbol{s})| \leq u \right\}.$$

In this special case, the partially stuck-at condition means that the output of the encoder is non-zero at each position of the support $\boldsymbol{\phi}$ *of* $\boldsymbol{s}$*.*

Definitions 4.2 and 4.3 describe error-correcting-and-defects-masking codes that can handle both partially defective cells and additional channel-added errors. In contrast, Definition 4.1 takes into account only the stuck-at cells and conveys codes for a masking-only scenario.

4.5.1 Constructions for at most $q-1$ partially stuck-at-1 Cells

We propose two code constructions over $\mathbb{F}_q$ for masking $u < q$ partially stuck-at-1 cells; one with linear codes using generator matrices in specific forms (cf. Section 2.3) and an alternate scheme using cyclic codes and polynomials (cf. Definition 2.5, Section 2.1.2, and Section 2.2.1).

Code Construction (less than q Partially Stuck Cells)

In this section, we present a coding scheme over $\mathbb{F}_q$ that can mask up to $q-1$ partially stuck cells and additionally can correct errors. We adapt the construction from [WY16], which allows to mask up to $q-1$ partially stuck-at-1 ($s_i = 1$ for all i) cells with only a single redundancy symbol, but cannot correct any substitution errors.

Construction 4.2. *Assume that there is an* $[n, k, d]_q$ *code* $\mathcal{C}$ *with a* $k \times n$ *generator matrix of the form*

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{G}_1 \\ \boldsymbol{G}_0 \end{bmatrix} = \begin{bmatrix} \boldsymbol{0}_{(k-1)\times 1} & \boldsymbol{I}_{k-1} & \boldsymbol{P}_{(k-1)\times(n-k)} \\ 1 & \boldsymbol{1}_{k-1} & \boldsymbol{1}_{n-k} \end{bmatrix},$$

where $\boldsymbol{I}_{k-1}$ *is the* $(k-1) \times (k-1)$ *identity matrix,* $\boldsymbol{P} \in \mathbb{F}_q^{(k-1)\times(n-k)}$*, and* $\boldsymbol{1}_\ell$ *is the all-one vector of length* ℓ*. From the code* $\mathcal{C}$*, a PSMC can be obtained, whose encoder and decoder are shown in Algorithm 3 and Algorithm 4.*

Theorem 4.2. *The coding scheme in Construction 4.2 is a* $(q-1, 1, \lfloor \frac{d-1}{2} \rfloor)$ *PSMC of length* n *and cardinality* q^{k-1}.

Algorithm 3: Encoding

```
Input:
  • Message: $\boldsymbol{m} = (m_0, m_1, \dots, m_{k-2}) \in \mathbb{F}_q^{k-1}$
  • Positions of partially stuck-at-1 cells: $\boldsymbol{\phi}$
1 Compute $\boldsymbol{w} = (w_1, w_2, \dots, w_{n-1}) = \boldsymbol{m} \cdot \boldsymbol{G}_1$
2 Find $v \in \mathbb{F}_q \setminus \{w_i \mid i \in \boldsymbol{\phi}\}$
3 Compute $\boldsymbol{c} = \boldsymbol{w} - v \cdot \boldsymbol{G}_0$
Output: Codeword $\boldsymbol{c} \in \mathbb{F}_q^n$
```

Algorithm 4: Decoding

```
Input:
  • Retrieve $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e}$ , $\boldsymbol{y} \in \mathbb{F}_q^n$
1 $\hat{\boldsymbol{c}} \leftarrow$ decode $\boldsymbol{y}$ in $\mathcal{C}$
2 $\hat{v} \leftarrow$ first entry of $\hat{\boldsymbol{c}}$
3 $\hat{\boldsymbol{w}} = (\hat{w}_0, \hat{w}_1, \cdots, \hat{w}_{n-1}) \leftarrow (\hat{\boldsymbol{c}} - \hat{v} \cdot \boldsymbol{G}_0)$
4 $\hat{\boldsymbol{m}} \leftarrow (\hat{w}_1, \dots, \hat{w}_{k-1})$
Output: Message vector $\hat{\boldsymbol{m}} \in \mathbb{F}_q^{k-1}$
```

Proof. To mask the partially stuck-at-1 positions, the codeword has to fulfill:

$$c_i \geq 1, \text{ for all } i \in \boldsymbol{\phi}. \tag{4.1}$$

Since $\mid \boldsymbol{\phi} \mid < q$, there is at least one value $v \in \mathbb{F}_q$ such that $w_i \neq v$, for all $i \in \boldsymbol{\phi}$. Thus, $c_i = (w_i - v) \neq 0$ and (4.1) is satisfied.

The decoder (Algorithm 4) gets $\boldsymbol{y}$, which is $\boldsymbol{c}$ corrupted by at most $\lfloor \frac{d-1}{2} \rfloor$ substitution errors (cf. Theorem 2.3 in Section 2.3.3). The decoder of $\mathcal{C}$ can correct these errors and obtain $\boldsymbol{c}$.

Due to the structure of $\boldsymbol{G}$, the first position of $\boldsymbol{c}$ equals $-v$. Hence, we can compute $\hat{\boldsymbol{w}} = \boldsymbol{w}$ (cf. Algorithm 4) and $\hat{\boldsymbol{m}} = \boldsymbol{m}$. □

Corollary 4.1. *If there is an* $[n, k, d]_q$ *code containing a word of weight* n*, then there is a* q*-ary* $(q-1, 1, \lfloor \frac{d-1}{2} \rfloor)$ *PSMC of length* n *and size* q^{k-1}.

To obtain a cyclic code, similar to [Hee83, Theorem 2], we can adapt Algorithms 3 and 4 of Construction 4.2 to directly operate on the generator polynomial of the code, which may be beneficial in practice. We will present this variant in Construction 4.3. For instance, any cyclic code whose generator polynomial is a divisor of $g_0(x) = 1 + x + x^2 + \cdots + x^{n-1}$ contains the all-one codeword. For BCH codes, this is the case if the defining set of the code does not contain 0. This gives an explicit family of codes whose parameters, for a specific choice of cyclotomic cosets, can be bounded by standard bounds on the minimum distance of cyclic codes such as the BCH bound.

Comparison to the Conventional Stuck Cell Scenario

Theorem 4.2 combines [Hee83, Theorem 1] and [WY16, Theorem 4] to provide a code construction that can mask partially stuck cells and correct errors. The required redundancy is a single symbol for masking plus the redundancy for the code generated by the upper part of $\boldsymbol{G}$, needed for the error correction. In comparison, [Hee83, Theorem 1] (see also Construction 4.1) requires at least

$$\min\{n-k \,:\, \exists\, [n,k,d]_q \text{ code with } d > u\} \geq u$$

redundancy symbols to mask u stuck cells, where the inequality follows directly from the Singleton bound.

In the following, we present Tables 4.1 and 4.2 to compare ternary cyclic codes of length $n = 8$ for masking partially stuck cells to mask stuck cells [Hee83], both with error correction.

The tables display that masking partially stuck cells requires less redundancy than masking stuck cells, both with and without additional error correction. The reason is that there is only one forbidden value in each partially stuck-at-1 cell, while there are $q-1$ forbidden values in each stuck-at cell. We provide the following remarks on Construction 4.2.

Remark 4.1. *The special case of Theorem 4.2 with $n < q$ was used in [Sol74] for constructing a $(q-1)$-ary error-correcting code from a q-ary Reed-Solomon code, which can be of interest if $q-1$ is not the power of a prime.*

Remark 4.2. *The code constructions in Theorem 4.2 and 4.3 also work over the ring of integers modulo q $(\mathbb{Z}/q\mathbb{Z})$ in which q is not necessarily a prime power, similar to the construction for $u < q$ in [WY16].*

Remark 4.3. *According to [WY16, Construction 3], it is possible to further decrease the required redundancy for masking u partially stuck-at-1 cells to $1 - \log_q \lfloor \frac{q}{u+1} \rfloor$. We can use the same strategy here. Let $z = \lfloor (\frac{q}{u+1}) \rfloor$. We choose disjoint sets $A_1, A_2, .., A_z$ of size $u+1$ in $\mathbb{F}_q$. As additional information, the encoder picks $j \in \{1, 2, \ldots, z\}$. In Step 2 of Algorithm 1, it selects v from A_j. As the decoder acquires v, it can obtain j as well.*

Table 4.1: Ternary Codes for *Partially* Stuck-at-1 Memory Cell for $n = 8$

Cardinality	Overall redundancy	u	t	Defining set D_c by (2.22)
3^7	1	2	0	$\{4\}$
3^6	2	2	0	$\{4\}$
3^5	3	2	0	$\{5, 7\}$
3^4	4	2	1	$\{4, 5, 7\}$
3^3	5	2	1	$\{1, 2, 3, 6\}$
3^2	6	2	1	$\{1, 2, 3, 4, 6\}$
3^2	6	2	1	$\{1, 3, 4, 5, 7\}$
3	7	2	1	$\{1, 2, 3, 5, 6, 7\}$

Table 4.2: Ternary Codes for Stuck-at Memory [Hee83] for $n = 8$

Cardinality	Overall Redundancy	u	t	The defining set D_c by (2.22)
3^7	1	1	0	$\{0\}$
3^6	2	1	0	$\{0\}$
3^5	3	1	0	$\{5, 7\}$
3^4	4	1	1	$\{0, 1, 3\}$
3^3	5	1	1	$\{1, 2, 3, 6\}$
3^2	6	1	2	$\{0, 1, 2, 3, 6\}$
3^2	6	2	1	$\{0, 1, 3\}$
3	7	2	1	$\{1, 2, 3, 6\}$

Variant of the Code Construction for less than q Partially Stuck Cells

This section provides an alternative of Construction 4.2 by generalizing the construction of [Hee83, Theorem 2]. We use the *partitioned cyclic codes* (see Definition 2.5 and Definition 2.6) from [Hee83] as basic idea, but we require only a single redundancy symbol $l = 1$ for the masking operation similar to [WY16, Theorem 4 and Algorithm 3]. Compared to Construction 4.2, Construction 4.3 directly implies a constructive strategy for choosing a cyclic code of a certain minimum distance that is directly applicable for memories with partially stuck cells that encounter substitution errors.

Construction 4.3 (Variant of Construction 4.2)**.** *Let* $u \leq \min\{n, q-1\}$*. Assume there is an* $[n, k, \geq 2t+1]_q$ *cyclic code* $\mathcal{C}$ *with a generator polynomial* $g(x)$ *of degree* $< n-k$ *that divides* $g_0(x) := 1 + x + x^2 + \cdots + x^{n-1}$*. Encoder and decoder are given in Algorithms 5 and 6.*

Theorem 4.3. *If $u \leq \min\{n, q-1\}$, Construction 4.3 provides an $(n, M = q^{k-1})_q$ $(u, 1, t)$ PSMC with redundancy of $n-k+1$ symbols.*

Algorithm 5: Encoding

```
Input:
    • Message: $m(x) \in \mathbb{F}_q[x]$ of degree deg $m(x) < k-1$
    • Positions of partially stuck cells: $\boldsymbol{\phi}$
1 $w(x) = w_0 + \cdots + w_{n-2}x^{n-2} \leftarrow m(x) \cdot g(x)$
2 Select $v \in \mathbb{F}_q \setminus \{w_i \mid i \in \boldsymbol{\phi}\}$.
3 $c(x) = w(x) - v \cdot g_0(x) \mod (x^n - 1)$
Output: Codeword $c(x) \in \mathbb{F}_q[x]$ of degree deg $c(x) \leq n-1$
```

Algorithm 6: Decoding

```
Input: Retrieve $y(x) = c(x) + e(x)$, where $e(x) \in \mathbb{F}_q[x]$ of degree deg
       $e(x) \leq n-1$ is the error polynomial
1 $\hat{c}(x) \leftarrow$ Decode $y(x)$ in the code generated by $g(x)$
2 $\hat{m}(x) \leftarrow \hat{c}(x) \mod g_0(x)$
Output: Message $\hat{m}(x) \in \mathbb{F}_q[x]$ of degree deg $\hat{m}(x) < k-1$
```

Proof. A cyclic code of length n contains the all-one word if and only if its generator polynomial $g(x)$ divides $g_0(x) = 1 + x + \cdots + x^{n-1}$. Thus, Construction 4.3 follows directly from Theorem 4.2, but with different encoding and decoding algorithms. Algorithm 5 shows the encoding process for the cyclic code construction. Step 1 in Algorithm 5 calculates $w(x)$ of degree deg $w(x) < n-1$. Since $u < q$, there is at least one $v \in \mathbb{F}_q$ such that all coefficients of $w(x)$, $w_i \in \mathbb{F}_q$, are unequal to v. Therefore, after Step 3, $c_{n-1} = -v$. The requirement for masking, see (4.1) is satisfied for $c(x)$ since $c_i = (w_i - v) \in \mathbb{F}_q \neq 0$.

Algorithm 6 decodes the retrieved polynomial $y(x)$. First, decode $y(x)$ in the code generated by $g(x)$. Second, the algorithm performs the unmasking process to find $\hat{m}(x)$. We obtain:

$$\hat{c}(x) = \hat{m}(x) \cdot g(x) - v \cdot g_0(x)$$
$$\hat{m}(x) = \frac{\hat{w}(x) \mod g_0(x)}{g(x)} = m(x). \qquad \square$$

Construction 4.3 provides an *explicit* cyclic construction that can mask $u < q$ cells and correct t errors (unique decoding as described in Section 2.3.3). If we use a BCH code in Construction 4.3, we can bound the minimum distance of the code $\mathcal{C}$ by the BCH bound. This is done in Tables 4.1 and 4.2.

Example 4.2. *In the error-free case in Construction 4.2 and Construction 4.3, single parity-check codes (SPC) over $\mathbb{F}_q$ (cf. Section 2.3.5) can directly be applied on a memory of n cells (cf. Chapter 3) to mask u partially stuck cells (cf. Section 3.4.3) less than q while storing $n-1$ information symbols since it needs only a single symbol as a redundancy. Table 4.3 exhibits that as the cardinality in the sixth row is q^{n-1}.* ▶

4.5.2 Constructions for more than $q-1$ partially stuck-at-1 Cells

The masking technique in the previous section only guarantees successful masking up to a number of $q-1$ partially stuck-at-1 cells. In this section, we present techniques to mask more than $q-1$ cells.

Depending on the values of the stored information in the partially stuck positions, Construction 4.2 may be able to mask more than $q-1$ cells. In the following section, we determine the probability that masking is possible for fixed partially stuck cell positions and randomly chosen information vectors.

Next, we propose two code constructions for simultaneous masking and error correction when $q \leq u < n$. One is based on the masking-only construction in [WY16, Construction 4] and the other is based on [WY16, Section VI], which are able to mask $u \geq q$ partially stuck positions, but cannot correct any errors. We generalize these constructions to be able to cope with errors. The latter construction may lead to larger code dimensions for a given pair (u, t), in a similar fashion as [WY16, Construction 5] improves upon [WY16, Construction 4]. Further, taking $t = 0$ it achieves larger codes sizes than [WY16, Construction 5] if the all-one word is in the code.

Probabilistic Masking

We determine the probability that masking is possible for $u \geq q$ partially stuck-at 1 cells stuck positions with the code constructions in Theorem 4.2 and Theorem 4.3. This *probabilistic masking* approach enables us to use the memory cells with a certain probability even if there are more than $q-1$ partially stuck cells.

Theorem 4.4. *Let $\boldsymbol{G}$ be as in Construction 4.2, and let $\boldsymbol{\phi} \subset [n]$ have size u. If the columns of $\boldsymbol{G}$ indexed by the elements in $\boldsymbol{\phi}$ are linearly independent, a uniformly drawn message from $\mathbb{F}_q^{k-1}$ results in a word $\boldsymbol{c}$ with $c_i \neq 0$ for all $i \in \boldsymbol{\phi}$ with probability*

$$\mathrm{P}(q,u) = 1 - \frac{\sum_{i=0}^{q-1}(-1)^i\binom{q}{i}(q-i)^u}{q^u}. \tag{4.2}$$

Proof. An appropriate value for v in Step 2 in Algorithm 3 cannot be found if and only if $\{w_i \mid i \in \boldsymbol{\phi}\} = \mathbb{F}_q$ which is true if and only if $f : \boldsymbol{\phi} \mapsto \mathbb{F}_q$ defined by $f_w(i) = w_i$ is a surjection. As is well-known (see e.g [LW01, Example 10.2], the number of surjections from a set of size u to a set of size q equals

$$\sum_{i=0}^{q-1} (-1)^i \binom{q}{i} (q-i)^u. \tag{4.3}$$

As the columns of $\boldsymbol{G}$ are independent, the vector $\boldsymbol{w}$ restricted to $\boldsymbol{\phi}$ is distributed uniformly on $\mathbb{F}_q^u$, and hence a word is not masked with probability equal to the expression from (4.3) divided by q^u. □

The following example illustrates that the probability that masking is successful can be quite large.

Example 4.3. *Let $q = 3$, $n = 8$, $n - k = 0$. The probability to mask $u = n - 1$ partially stuck-at-1 memory cells is $\mathrm{P}(3, 7) = 0.17$. This ratio is 0.77 if $u = q$ and clearly it is 1 if $u < q$.* ▶

Remark 4.4. *The assumption in Theorem 4.4 that the columns of $\boldsymbol{G}$ indexed by the partially stuck positions are linearly independent is fulfilled for most codes with high probability if $u \leq k - 1$, especially if $u \ll k - 1$. For dependent columns, it becomes harder to count the number of intermediate codewords $\boldsymbol{w}$ that do not cover the entire alphabet since w_i for all $i \in \boldsymbol{\phi}$ is not uniformly distributed over $\mathbb{F}_q^u$.*

Code Construction (up to $q + d_0 - 3$ Partially Stuck Cells)

We recall that [WY16, Construction 4] can mask more than $q - 1$ partially stuck-at-1 cells and it is a generalization of the all-one vector construction [WY16, Theorem 4]. Hence, replacing the $\mathbf{1}_n$ vector in Theorem 4.2 by a parity-check matrix as in [WY16, Construction 4] allows masking of q or more partially stuck-at 1 cells, and correct t errors (i.e., bounded minimum distance decoding in Section 2.3.3).

Construction 4.4. *Suppose that there is an $[n, k, d]_q$ code $\mathcal{C}$ with a $k \times n$ generator matrix of the following form:*

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{G}_1 \\ \boldsymbol{H}_0 \end{bmatrix}$$

where $\boldsymbol{H}_0 \in \mathbb{F}_q^{l \times n}$ is a parity-check matrix of an $[n, n-l, d_0]$ code $\mathcal{C}_0$. From the code $\mathcal{C}$, a PSMC can be obtained, whose encoder and decoder are shown in Algorithm 7 and Algorithm 8.

Theorem 4.5. *The coding scheme in Construction 4.4 is a $(d_0+q-3, 1, \lfloor\frac{d-1}{2}\rfloor)$ PSMC of length n and cardinality q^{k-l}.*

Algorithm 7: Encoding

```
Input:
    • Message: $\boldsymbol{m} \in \mathbb{F}_q^{k-l}$
    • Positions of partially stuck-at-1 cells: $\boldsymbol{\phi}$
1 Compute $\boldsymbol{w} = (w_1, w_2, \dots, w_{n-1}) = \boldsymbol{m} \cdot \boldsymbol{G}_1$
2 Find $\left\{ \boldsymbol{z} = (z_0, \dots, z_{l-1}) \in \mathbb{F}_q^l \;\middle|\; (\boldsymbol{w} + \boldsymbol{z}\boldsymbol{H}_0)_i \neq 0, \text{ for all } i \in \boldsymbol{\phi} \right\}$.
3 Compute $\boldsymbol{c} = \boldsymbol{w} + \boldsymbol{z} \cdot \boldsymbol{H}_0$
Output: Codeword $\boldsymbol{c} \in \mathbb{F}_q^n$
```

Algorithm 8: Decoding

```
Input: $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e} \in \mathbb{F}_q^n$
1 $\hat{\boldsymbol{c}} \leftarrow$ decode $\boldsymbol{y}$ in the code $\mathcal{C}$
2 Determine $\hat{\boldsymbol{m}} \in \mathbb{F}_q^{k-l}$ and $\hat{\boldsymbol{z}} \in \mathbb{F}_q^l$ such that $\hat{\boldsymbol{c}} = \hat{\boldsymbol{m}}\boldsymbol{G}_1 + \hat{\boldsymbol{z}}\boldsymbol{H}_0$.
Output: Message vector $\hat{\boldsymbol{m}} \in \mathbb{F}_q^{k-l}$
```

Proof. For the masking part, the proof is a simple modification of [WY16, Theorem 7]. In Section 4.6, we give a full proof of Proposition 4.2, which generalizes Construction 4.4. The error correction part of the proof follows the proof of Theorem 4.2. □

The gain of Theorem 4.5 in the number of partially stuck cells that can be masked comes at the cost of larger redundancy. However, the redundancy is still smaller than the redundancy of the construction for masking stuck-at cells and error correction in [Hee83]. In particular, let $\mathcal{C}$ be an $[n, k, d \geq 2t + 1]$ code containing an $[n, l]_q$ subcode $\mathcal{C}_0$ for which $\mathcal{C}_0^\perp$ has minimum distance d_0. With Theorem 4.5, we obtain a $(d_0 + q - 3, 1, \lfloor \frac{d-1}{2} \rfloor)$ PSMC of length n and cardinality q^{k-l}. The construction in [Hee83] yields a coding scheme with equal cardinality, allowing for masking up to $d_0 - 1$ fully stuck cells and correcting $\lfloor \frac{d-1}{2} \rfloor$ errors (cf. Theorem 2.3 in Section 2.3.3) since the minimum distance d_1 defined in [Hee83] for the error correction capability of the code is d in our notation. Hence, exactly $q - 2$ more cells that are partially-stuck-at levels 1 than classical stuck cells can be masked.

Example 4.4. *We apply Construction 4.4 to mask up to $u = 4$ partially stuck cells over $\mathbb{F}_4$ and $\boldsymbol{m} \in \mathbb{F}_4^9$. Let α be a primitive element in $\mathbb{F}_{16}$ (cf. Definition 2.3) and let $\mathcal{C}$ be the $[15, 12, 3]_4$ code with zeros α^0 and α^1. Let $\mathcal{C}_0$ be the $[15, 3]$ subcode of $\mathcal{C}$ be the BCH code with zeros $\{\alpha^i \mid 0 \leq i \leq 14\} \setminus \{\alpha^5, \alpha^6, \alpha^9\}$. As $\mathcal{C}_0^\perp$ is equivalent to the $[15, 12, 3]_4$ code with zeros $\alpha^5, \alpha^6, \alpha^9$, it has minimum distance $d_0 = 3$. Hence, we obtain a $(4, 1, 1)$ PSMC code of cardinality 4^9.* ▶

Code Construction (up to $2^{\mu-1}(d_0+1)-1$ Partially Stuck Cells)

We generalize [WY16, Section VI] to be able to cope with errors. Unlike [WY16, Section VI] that could be over any prime power q, the following code construction works over the finite field $\mathbb{F}_q$ where $q = 2^\mu$ in order to describe a 2^μ-ary partially stuck cells code construction. This is because binary sub-field subcodes that are required in this construction are not linear subspace for codes over any prime power q. We denote by $\beta_0 = 1, \beta_1, \dots, \beta_{\mu-1}$ a basis of $\mathbb{F}_{2^\mu}$ over $\mathbb{F}_2$. That is, any element $a \in \mathbb{F}_{2^\mu}$ can be uniquely represented as $a = \sum_{i=0}^{\mu-1} a_i\beta_i$ where $a_i \in \mathbb{F}_2$ for all i (see Section 2.2). In particular, $a \in \mathbb{F}_2$ if and only if $a_1 = \cdots = a_{\mu-1} = 0$. This is a crucial property of $\mathbb{F}_{2^\mu}$ that we will use in Construction 4.5.

Construction 4.5. *Let $\mu > 1$. Suppose $\boldsymbol{G}$ is a $k \times n$ generator matrix of an $[n,k,d]_{2^\mu}$ code $\mathcal{C}$ of the form*

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{H}_0 \\ \boldsymbol{G}_1 \\ \boldsymbol{x} \end{bmatrix} \tag{4.4}$$

where

1. *$\boldsymbol{H}_0 \in \mathbb{F}_2^{l\times n}$ is a parity-check matrix of an $[n, n-l, d_0]_2$ code $\mathcal{C}_0$,*
2. *$\boldsymbol{G}_1 \in \mathbb{F}_{2^\mu}^{(k-l-1)\times n}$,*
3. *$\boldsymbol{x} \in \mathbb{F}_{2^\mu}^{1\times n}$ has Hamming weight n.*

From the code $\mathcal{C}$, a PSMC can be obtained, whose encoder and decoder are shown in Algorithm 9 and Algorithm 10.

Theorem 4.6. *The coding scheme in Construction 4.5 is a 2^μ-ary $(2^{\mu-1}d_0-1, 1, \lfloor\frac{d-1}{2}\rfloor)$ PSMC of length n and cardinality $2^{\mu(k-l-1)}2^{l(\mu-1)}$.*

Proof. Let $\boldsymbol{\phi} \subset [n]$ have size $u \leq 2^{\mu-1}d_0 - 1$.
We first show the existence of z from Step 1. For each $i \in \boldsymbol{\phi}$, we have that $\boldsymbol{x}_i \neq 0$, so there are exactly two elements $z \in \mathbb{F}_{2^\mu}$ such that $(\boldsymbol{m}' \cdot \boldsymbol{H}_0 + \boldsymbol{m} \cdot \boldsymbol{G}_1)_i + z\boldsymbol{x}_i \in \mathbb{F}_2$. As a result,

$$2u = 2|\boldsymbol{\phi}| = \mid \{(i,z) \in \boldsymbol{\phi} \times \mathbb{F}_{2^\mu} \mid (\boldsymbol{m}' \cdot \boldsymbol{H}_0 + \boldsymbol{m} \cdot \boldsymbol{G}_1)_i + z\boldsymbol{x}_i \in \mathbb{F}_2\} \mid .$$

As $u < 2^{\mu-1}d_0$, there is a $z \in \mathbb{F}_{2^\mu}$ such that the condition in Step 1 is satisfied.

As $\boldsymbol{H}_0$ is the parity-check matrix of a code with minimum distance d_0, any $d_0 - 1$ columns of $\boldsymbol{H}_0$ are independent, so an appropriate $\boldsymbol{\gamma}$ exists. Now we show that $c_i \neq 0$ for all $i \in \boldsymbol{\phi}$. Indeed, if $\boldsymbol{w}_i \notin \mathbb{F}_2$, then $\boldsymbol{c}_i = \boldsymbol{w}_i + (\boldsymbol{\gamma}\boldsymbol{H}_0)_i \in \{\boldsymbol{w}_i, \boldsymbol{w}_i + 1\}$, so $\boldsymbol{c}_i \notin \mathbb{F}_2$. By Step 2 in Algorithm 9, for $\boldsymbol{w}_i \in \mathbb{F}_2$, we have that $\boldsymbol{c}_i = 1$. Hence, for all $i \in \boldsymbol{\phi}$, $\boldsymbol{c}_i$ is either 1 or is in $\mathbb{F}_{2^\mu} \setminus \mathbb{F}_2$, i.e., $c_i \neq 0$.

```
Algorithm 9: Encoding ($\boldsymbol{m}; \boldsymbol{m}'; \boldsymbol{\phi}$)
Input:
  • Message:
    $(\boldsymbol{m}', \boldsymbol{m}) \in \mathcal{F}^l \times \mathbb{F}_{2^\mu}^{k-l-1}$, where
    $\mathcal{F} = \{\sum_{i=1}^{\mu-1} x_i\beta_i \mid (x_1, \dots, x_{\mu-1}) \in \mathbb{F}_2^{\mu-1}\}$.
  • Positions of partially stuck-at-1 cells: $\boldsymbol{\phi}$
  • Notations introduced in Construction 4.5.
1 $\boldsymbol{w} \leftarrow \boldsymbol{m}' \cdot \boldsymbol{H}_0 + \boldsymbol{m} \cdot \boldsymbol{G}_1 + z \cdot \boldsymbol{x}$ where $z \in \mathbb{F}_{2^\mu}$ is chosen such that
    $|\{i \in \boldsymbol{\phi} \mid w_i \in \mathbb{F}_2\}| \leq d_0 - 1$.
2 Choose $\boldsymbol{\gamma} \in \mathbb{F}_2^l$ such that $(\boldsymbol{\gamma}\boldsymbol{H}_0)_i = 1 - \boldsymbol{w}_i$ for all $i \in \boldsymbol{\phi}$ for which $\boldsymbol{w}_i \in \mathbb{F}_2$.
Output: $\boldsymbol{c} = \boldsymbol{w} + \boldsymbol{\gamma} \cdot \boldsymbol{H}_0 \in \mathcal{C}$
```

```
Algorithm 10: Decoding
Input:
  • $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e} \in \mathbb{F}_{2^\mu}^n$, where $\boldsymbol{c}$ is a valid output of Algorithm 9 and $\boldsymbol{e}$ is an error of
    Hamming weight at most $t$.
  • Notations introduced in Construction 4.5.
1 $\hat{\boldsymbol{c}} \leftarrow$ decode $\boldsymbol{y}$ in the code $\mathcal{C}$
2 Obtain $\boldsymbol{a} \in \mathbb{F}_{2^\mu}^l, \hat{\boldsymbol{m}} \in \mathbb{F}_{2^\mu}^{k-l-1}, \hat{z} \in \mathbb{F}_{2^\mu}$ such that $\hat{\boldsymbol{c}} = \boldsymbol{a}\boldsymbol{H}_0 + \hat{\boldsymbol{m}}\boldsymbol{G}_1 + \hat{z}\boldsymbol{x}$.
3 Obtain $\hat{\boldsymbol{m}'} \in \mathcal{F}^{k-l-1}$ and $\hat{\boldsymbol{\gamma}} \in \mathbb{F}_2^{k-l-1}$ such that $\boldsymbol{a} = \hat{\boldsymbol{m}'} + \hat{\boldsymbol{\gamma}}$.
Output: $(\hat{\boldsymbol{m}}, \hat{\boldsymbol{m}}')$
```

Decoding: As $\boldsymbol{c} \in \mathcal{C}$, $\hat{\boldsymbol{c}} = \boldsymbol{c}$. As $\boldsymbol{G}$ has full rank (cf. Section 2.3), and

$$\boldsymbol{c} = (\boldsymbol{m}' + \boldsymbol{\gamma})\boldsymbol{H}_0 + \boldsymbol{m}\boldsymbol{G}_1 + z\boldsymbol{x},$$

it holds that $\boldsymbol{a} = \hat{\boldsymbol{m}'} + \hat{\boldsymbol{\gamma}}$, $\hat{\boldsymbol{m}} = \boldsymbol{m}$ and $\hat{z} = z$. As $\hat{\boldsymbol{m}'} \in \mathcal{F}^l$ and $\hat{\boldsymbol{\gamma}} \in \mathbb{F}_2^l$, we can retrieve $\hat{\boldsymbol{m}'} = \boldsymbol{m}'$ from $\boldsymbol{a} = \hat{\boldsymbol{m}'} + \hat{\boldsymbol{\gamma}}$. □

Remark 4.5. *Construction 4.5 is a generalized and improved version of our construction presented in [APW20].*

We demonstrate next two minor extensions of Theorem 4.6 for the special case that $\boldsymbol{x}$ is the all-one vector.

Proposition 4.1. *If $\boldsymbol{x}$ is the all-one vector in Theorem 4.6, then the coding scheme in Construction 4.5 can be modified to produce a 2^μ-ary $(2^{\mu-1}d_0 - 1, 1, \lfloor \frac{d-1}{2} \rfloor)$ PSMC* *of length n and cardinality* $2 \times 2^{\mu(k-l-1)}2^{l(\mu-1)}$.

Proof. For $\boldsymbol{x} = \mathbf{1}$, if $\boldsymbol{m}'\boldsymbol{H}_0 + \boldsymbol{m}\boldsymbol{G}_1 + z\mathbf{1}$ has at most $d_0 - 1$ binary entries, then so has $\boldsymbol{m}'\boldsymbol{H}_0 + \boldsymbol{m}\boldsymbol{G}_1 + (z+1)\mathbf{1}$. Hence, there is a $z_0 \in \mathcal{F}$ such that $\boldsymbol{w} + z_0\mathbf{1}$ has at most $d_0 - 1$ binary entries, and we can encode

$$\boldsymbol{w} = \boldsymbol{m}'\boldsymbol{H}_0 + \boldsymbol{m}\boldsymbol{G}_1 + (z_0 + \zeta)\mathbf{1},$$

where $\zeta \in \{0,1\}$ is an additional message bit so that the cardinality from Theorem 4.6 is doubled. As $z_0 \in \mathcal{F}$ and $\zeta \in \{0,1\}$, the pair (z_0, ζ) can be retrieved from $z_0 + \zeta$. □

Construction 4.5.A (Extension of Construction 4.5)**.** *Let $\boldsymbol{G}$ be a $k \times n$ generator matrix of an $[n,k,d]_{2^\mu}$ code $\mathcal{C}$ of the form*

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{H}_0 \\ \boldsymbol{G}_1 \\ \mathbf{1} \end{bmatrix}, \text{ where}$$

1) $\mathbf{1}$ is the all-one vector of length n
2) $\boldsymbol{G}_1 \in \mathbb{F}_q^{k-l-1\times n}$
3) $\begin{bmatrix} \boldsymbol{H}_0 \\ \mathbf{1} \end{bmatrix}$ is the parity-check matrix of an $[n, n-l-1, d_e]_2$ code.

Theorem 4.6.A. *If the conditions of Construction 4.5.A hold, then Construction 4.5 can be modified to produce a 2^μ-ary $(2^{\mu-1}d_e, 1, \lfloor\frac{d-1}{2}\rfloor)$ PSMC of length n and cardinality $2^{\mu(k-l-1)}2^{l(\mu-1)}$.*

Proof. In Step 1 of Algorithm 9, the encoder determines z such that $|\ \{i \in \boldsymbol{\phi} \mid w_i \in \mathbb{F}_2\}\ | \ \leq d_e - 1$; the existence of such a z is proved as in the proof of Theorem 4.5. Next, the encoder determines $\boldsymbol{\gamma} \in \{0,1\}^l$ and $\gamma_0 \in \{0,1\}$ such that

$$\boldsymbol{v} = (\boldsymbol{\gamma}, \gamma_0) \cdot \begin{bmatrix} \boldsymbol{H}_0 \\ \mathbf{1} \end{bmatrix}$$

is such that $v_i = 1 - w_i$ for all $i \in \boldsymbol{\phi}$ for which $w_i \in \{0,1\}$. The encoding output $\boldsymbol{c} = \boldsymbol{v} + \boldsymbol{w} = (\boldsymbol{m}' + \boldsymbol{\gamma})\boldsymbol{H}_0 + \boldsymbol{m}\boldsymbol{G}_1 + (z_0 + \gamma_0)\mathbf{1}$ thus is in $\mathcal{C}$ and has no zeros in the positions of $\boldsymbol{\phi}$.

In decoding, from $\boldsymbol{c}$ both $(\boldsymbol{m}' + \boldsymbol{\gamma})$ and $\boldsymbol{m}$ can be retrieved, and so, as $\boldsymbol{m}' \in \mathcal{F}^l$ and $\boldsymbol{\gamma} \in \{0,1\}^l$, $\boldsymbol{m}'$ can be retrieved as well. □

Proposition 4.1 doubles the size of the PSMC as compared to Theorem 4.6 (by using ζ as additional message bit), while masking the same number of partially stuck-at-errors and correcting the same number of substitution errors. Theorem 4.6.A, as compared to Theorem 4.6, results in a PSMC of the same size and error correction capabilities, but increases the number of cells that can be masked from $2^{\mu-1}d_0 - 1$ to $2^{\mu-1}d_e - 1$. If d_0 is odd, then this increment is at least $2^{\mu-1}$.

Now, we come up with an example using *nested* BCH codes (see Section 2.3.6), allowing to store more symbols compared to Theorem 4.6 for the same code parameters.

Example 4.5. *Let α be a primitive 15th root of unity in $\mathbb{F}_{16}$ (refer to Definition 2.3 and Remark 2.1), and let $\mathcal{C}$ be the $[15, 12, 3]_4$ BCH code with zeros α^5, α^6 and α^9 (see Definition 2.8). Let the $[15, 4]_2$ subcode $\mathcal{C}_0^{\perp}$ of $\mathcal{C}$ (cf. Definition 2.4) be defined as*

$$\mathcal{C}_0^{\perp} = \left\{ (x_0, \ldots, x_{14}) \in \mathbb{F}_2^{15} \;\middle|\; \sum_{i=0}^{14} x_i \alpha^{ij} = 0 \textit{ for } j \in \{0, \ldots, 14\} \setminus \{7, 11, 13, 14\} \right\}.$$

As $\mathbf{1} \in \mathcal{C} \backslash \mathcal{C}_0^{\perp}$, the code $\mathcal{C}$ has a generator matrix of the form given in Construction 4.5, namely

$$\boldsymbol{G}' = \begin{bmatrix} \boldsymbol{H}_0 \\ \boldsymbol{G}_1 \\ \boldsymbol{x} \end{bmatrix},$$

where $\boldsymbol{H}_0$ is a generator matrix for $\mathcal{C}_0^{\perp}$ (i.e., $\boldsymbol{H}_0$ is obtained by (2.8) in Section 2.3.4) and $\boldsymbol{G}_1$ has $12 - 4 - 1 = 7$ rows (i.e., $\boldsymbol{G}_1$ is obtained by (2.10) in Section 2.3.4). The code $\mathcal{C}_0 = (\mathcal{C}_0^{\perp})^{\perp}$ is equivalent to the $[15, 11]_2$ BCH code with zeros $\alpha^7, \alpha^{11}, \alpha^{13}$ and α^{14}. As this BCH code has two consecutive zeros, its minimum distance (and hence the minimum distance of $\mathcal{C}_0$) is at least 3.

We stipulate that $\alpha^4 = \alpha + 1$ to obtain explicit $\boldsymbol{G}'$ as below,

$$\boldsymbol{G}' = \begin{pmatrix}
1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\
\omega & \omega & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & \omega & \omega & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \omega & \omega & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \omega & \omega & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \omega & \omega & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \omega & \omega & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \omega & \omega & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{pmatrix},$$

where $\mathbb{F}_4$ has elements $\{0, 1, \omega, \omega^2\}$ with $\omega = \alpha^5$. Note that the top row of $\boldsymbol{H}_0$ corresponds to the generator polynomial for $\mathcal{C}_0^{\perp}$, and the top row of $\boldsymbol{G}_1$ corresponds to the coefficients of $(x + \alpha^5)(x + \alpha^6)(x + \alpha^9)$ which is the generator polynomial of $\mathcal{C}$. Application of Proposition 4.1 yields a $(5, 1, 1)$ PSMC over $\mathbb{F}_{2^2}$ of length 15 and size $2 \times 4^7 2^4 = 2^{19}$, whereas application of Construction 4.5.A gives a $(7, 1, 1)$ PSMC over $\mathbb{F}_{2^2}$ with cardinality $2^{2(7+4)-4} = 2^{18}$. Note that application of Construction 4.5 yields a $(5, 1, 1)$ PSMC over $\mathbb{F}_{2^2}$ of length 15 and size $4^7 2^4 = 2^{18}$.

Finally, we note that application of Theorem 4.5 to $\mathcal{C}$ *yields a* $(4,1,1)$ *PSMC of size* 4^8*, which has worse parameters than the three PSMC mentioned before.* ▶

Example 4.5 clearly shows that for the same code parameters, Construction 4.5, Proposition 4.1 and Construction 4.5.A significantly improve upon Construction 4.4.

Remark 4.6. *For masking only, choose* $n-k=0$ *in Construction 4.5 and therefore,*

$$\boldsymbol{G}_1 = \begin{bmatrix} \mathbf{0}_{(n-l-1)\times(l+1)} & \boldsymbol{I}_{(n-l-1)} & \mathbf{0}_{(n-l-1)\times 1} \end{bmatrix},$$

and we can store $n-l-1$ *information symbols. Thus, Proposition 4.1 for masking only improves upon [WY16, Construction 5]. For example if* $l=4$*, then* $n-l-1=10$ *in [WY16, Example 7] and the size of the code is* $2^{2(n-l-1)} \cdot 2^l = 2^{24}$*, while* $n-l-1=10$ *in Proposition 4.1 for* $\boldsymbol{x}=\mathbf{1}$ *and the cardinality is* $2 \cdot 2^{2n-l-2} = 2^{25}$*.*

We summarize in Table 4.3 our constructions and compare them with some of the previous works, namely with the construction for masking classical stuck cells in [Hee83] and constructions for partially stuck cells without errors in [WY16]. Table 4.3 clearly shows that more information can be stored with partially stuck-at errors than with classical stuck cells.

Table 4.3: Comparison among [Hee83], [WY16], and this work. We denote by d the minimum distance required to correct errors and d_0 to mask (partially) stuck cells. A positive integer $\mu > 1$ is defined in Construction 4.5. Other Notations: See Section 2.1.

	(Partially) Stuck Cells u	**Distance** d_0	**Errors** $\lfloor \frac{d-1}{2} \rfloor$	**Redundancy**	**Cardinality**
Construction 4.2	$\leq q-1$	irrelevant	Yes	$n-k+1$	q^{k-1}
Construction 4.4	$\leq n$	$\geq u-q+3$	Yes	$n-k+l$	q^{k-l}
Construction 4.5	$\leq n$	$\geq \lfloor \frac{2u}{2^\mu} \rfloor + 1$	Yes	$n-k+1+\frac{l}{\mu}$	$q^{k-1-\frac{l}{\mu}}$
Proposition 4.1	$\leq n$	$\geq \lfloor \frac{2u}{2^\mu} \rfloor + 1$	Yes	$n-k+1+\frac{l-1}{\mu}$	$q^{k-1+\frac{1-l}{\mu}}$
Construction 4.5.A	$\leq n$	$\geq \lfloor \frac{2u}{2^\mu} \rfloor$ if d_0 is odd	Yes	$n-k+1+\frac{l}{\mu}$	$2^{2(k-1-\frac{l}{\mu})}$
[WY16, Construction 2]	$\leq q-1$	irrelevant	No	1 (since $n-k=0$)	q^{n-1}
[WY16, Construction 4]	$\leq n$	$\geq u-q+3$	No	l (since $n-k=0$)	q^{n-l}
[WY16, Construction 5]	$\leq n$	$\geq \lfloor \frac{2u}{q} \rfloor + 1$	No	$1+l(1-\log_q \lfloor \frac{q}{2} \rfloor)$ (since $n-k=0$), and $1+\frac{l}{\mu}$ (for $q=2^\mu$)	$q^{n-1-l(1-\log_q \lfloor \frac{q}{2} \rfloor)}$, and $2^{2(n-1-\frac{l}{\mu})}$ (for $q=2^\mu$)
Proposition 4.1 (masking only)	$\leq n$	$\geq \lfloor \frac{2u}{2^\mu} \rfloor + 1$	No	$1+\frac{l-1}{\mu}$ (since $n-k=0$)	$2^{2(n-1-\frac{l-1}{\mu})}$
[Hee83, Theorem 1]	$\leq n$	$\geq u+1$	Yes	$n-k+l$	q^{k-l}

4.6 Generalization to Arbitrary Partially Defective Levels

So far, we have considered the *important* case for $s_i = 1$ for all $i \in \boldsymbol{\phi}$ (i.e., the erasing state is avoided see Sections 3.3.1, 3.3.2 and 3.4.3). In this section, we present error correction and masking code constructions that can mask partially stuck cells at any level $\boldsymbol{s}_i$ and correct errors additionally.

4.6.1 Generalization of the Code Construction (less than q Partially Stuck Cells)

Here, we give only the main theorem without adding the exact encoding and decoding processes because it follows directly from Construction 4.2.

Theorem 4.7 (Generalization of Theorem 4.2)**.** *Let* $\Sigma = \{\boldsymbol{s} \in \mathbb{F}_q^n \mid \sum_{i=0}^{n-1} s_i \leq q-1\}$. *Assume there is an* $[n,k,d]_q$ *code* $\mathcal{C}$ *of a generator matrix as specified in Theorem 4.2. Then there exists a* $(\Sigma, \lfloor \frac{d-1}{2} \rfloor)$ *PSMC over* $\mathbb{F}_q$ *of length* n *and cardinality* q^{k-1}.

Proof. We follow the generalization for the masking partially-stuck-at any arbitrary levels in [WY16, Theorem 10]. Hence, for $\boldsymbol{s} \in \Sigma$, we modify Step 2 in Algorithm 3 such that $w_i - v \geq s_i$ for all $i \in [n]$. Such a v exists as each cell partially-stuck-at level s_i excludes s_i values for v, and $\sum_{i=0}^{n-1} s_i < q$. The rest of the encoding steps and the decoding process are analogous to Algorithms 3 and 4. As the output from the encoding process is a codeword, we can correct $\lfloor \frac{d-1}{2} \rfloor$ errors (see Theorem 2.3 in Section 2.3.3). □

4.6.2 Generalization of the Code Construction (up to $q + d_0 - 3$ Partially Stuck Cells)

In the following, we generalize Construction 4.4 to arbitrary $\boldsymbol{s}$ stuck levels.

Generalization Proposition to Arbitrary Stuck Levels

Proposition 4.2 (Generalization of Construction 4.4)**.** *Let*

$$\Sigma = \left\{ \boldsymbol{s} \in \mathbb{F}_q^n \;\middle|\; \min \left\{ \sum_{i \in \Psi} s_i \;\middle|\; \Psi \subseteq [n], |\,\Psi\,| = n - d_0 + 2 \right\} \leq q - 1 \right\}$$

then the coding scheme in Construction 4.4 can be modified to produce a $(\Sigma, \lfloor \frac{d-1}{2} \rfloor)$ *PSMC of length* n *and size* q^{k-l}.

Proof. To avoid cumbersome notation, we assume without loss of generality that $s_0 \geq s_1 \geq \cdots \geq s_{n-1}$. As the $d_0 - 2$ leftmost columns of $\boldsymbol{H}_0$ are independent, there is an invertible $\boldsymbol{T} \in \mathbb{F}_q^{l \times l}$ such that the matrix $\boldsymbol{Y} = \boldsymbol{T}\boldsymbol{H}_0$ has the form

$$\boldsymbol{Y} = \begin{bmatrix} I_{d_0-2} & \boldsymbol{A} \\ \boldsymbol{0} & \boldsymbol{B} \end{bmatrix},$$

where I_{d_0-2} is the identity matrix of size $d_0 - 2$, $\boldsymbol{0}$ denotes the $(l - d_0 + 2) \times (d_0 - 2)$ all-zero matrix, $\boldsymbol{A} \in \mathbb{F}_q^{(d_0-2)\times(n-d_0+2)}$ and $\boldsymbol{B} \in \mathbb{F}_q^{(l-d_0+2)\times(n-d_0+2)}$. As $\boldsymbol{T}$ is invertible, and any $d_0 - 1$ columns of $\boldsymbol{H}_0$ are independent, any $d_0 - 1$ columns of $\boldsymbol{Y}$ are independent as well.

For $0 \leq i \leq l - 1$, we define

$$L_i = \{j \in [n] \mid Y_{i,j} \neq 0 \text{ and } Y_{m,j} = 0 \text{ for } m > i\}. \tag{4.5}$$

Clearly, $L_0, \ldots, L_{l-1}$ are pairwise disjoint. Moreover, for each $j \in \{d_0-2, d_0-1, \ldots, n-1\}$, column j of $\boldsymbol{Y}$ is independent from the $(d_0 - 2)$ leftmost columns of $\boldsymbol{Y}$, and so there is an $i \geq d_0 - 2$ such that $Y_{i,j} \neq 0$. Consequently,

$$\bigcup_{i=d_0-2}^{l-1} L_i = \{d_0 - 2, \ldots, n - 1\}. \tag{4.6}$$

By combining (4.6) and the form of $\boldsymbol{Y}$, we infer that

$$L_k = \{k\} \text{ for all } k \in [d_0 - 2]. \tag{4.7}$$

Let $\boldsymbol{w} \in \mathbb{F}_q^n$ be the vector to be masked, i.e. the vector after Step 1 in Algorithm 7. The encoder successively determines the coefficients $z_0, \ldots, z_{l-1}$ of $\boldsymbol{z} \in \mathbb{F}_q^l$ such that $\boldsymbol{w} + \boldsymbol{z}\boldsymbol{Y} \geq \boldsymbol{s}$, as follows.
For $j \in [d_0 - 2]$, the encoder sets $z_j = s_j - w_j$.
Now let $d_0 - 2 \leq i \leq l - 1$ and assume that $z_0, \ldots, z_{i-1}$ have been obtained such that

$$w_j + \sum_{k=0}^{i-1} z_k Y_{k,j} \geq s_j \text{ for all } j \in \bigcup_{k=0}^{i-1} L_k. \tag{4.8}$$

It follows from combination of (4.7) and the choice of $z_0, \ldots, z_{d_0-3}$ that (4.8) is satisfied for $i = d_0 - 2$.

For each $j \in L_i$, we define F_j as

$$F_j = \left\{ x \in \mathbb{F}_q \;\middle|\; w_j + \sum_{k=0}^{i-1} z_k Y_{k,j} + x Y_{i,j} < s_j \right\}.$$

Clearly, $|F_j| = s_j$ as $Y_{i,j} \neq 0$, and so

$$\Big|\bigcup_{j\in L_i} F_j\Big| \leq \sum_{j\in L_i} \big|F_j\big| = \sum_{j\in L_i} s_j \leq \sum_{j=d_0-2}^{n-1} s_j \leq q-1,$$

where the last inequality follows from the assumption of Σ in the proposition statement and the ordering of the components of $\boldsymbol{s}$. Hence, $\bigcup_{j\in L_i} F_j \neq \mathbb{F}_q$. The encoder chooses $z_i \in \mathbb{F}_q \setminus \bigcup_{j\in L_i} F_j$. We claim that

$$w_j + \sum_{k=0}^{i} z_k Y_{k,j} \geq s_j \text{ for all } j \in \bigcup_{k=0}^{i} L_k. \tag{4.9}$$

For $j \in L_i$, (4.9) follows from the definition of F_j. For $j \in \bigcup_{k=0}^{i-1} L_k$, (4.9) follows from (4.8) and the fact that $Y_{i,j} = 0$.

By using induction on i, we infer that

$$w_j + \sum_{k=0}^{l-1} z_k Y_{k,j} \geq s_j \text{ for all } j \in \cup_{k=0}^{l-1} L_k = [n]. \tag{4.10}$$

That is, with $\boldsymbol{z} = (z_0, \ldots, z_{l-1})$, we have that $\boldsymbol{w} + \boldsymbol{z}\boldsymbol{Y} \geq \boldsymbol{s}$. As $\boldsymbol{Y} = \boldsymbol{T}\boldsymbol{H}_0$, it follows that $\boldsymbol{z} := \boldsymbol{z}\boldsymbol{T}$ is such that

$$\boldsymbol{w} + \boldsymbol{z}\boldsymbol{H}_0 \geq \boldsymbol{s}.$$

The decoding process remains as in Algorithm 8. □

An Alternative Proof of the Generalization Proposition

We give an alternative *non-constructive* proof for Proposition 4.2 in Appendix A.4.

Remark 4.7. *The proof of Proposition 4.2 shows that $(d_0 - 2)$ cells can be set to any desired value, while the remaining $(n - d_0 + 2)$ cells can be made to satisfy the partial stuck-at conditions, provided that the sum of the stuck-at levels in these $(n - d_0 + 2)$ cells is less than q.*

Example 4.6. *Let $\mathcal{C}$ be the $[13, 10, 3]_3$ generated by*

$$\boldsymbol{G} = \begin{bmatrix} \boldsymbol{G}_1 \\ \boldsymbol{H}_0 \end{bmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 2 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 2 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 2 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 2 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 & 0 & 0 & 1 & 1 & 2 & 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 \end{pmatrix} \in \mathbb{F}_3^{10\times 13}$$

Let $\mathcal{C}_0$ be the code with parity check matrix $\boldsymbol{H}_0$. Let

$$\boldsymbol{s} = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \in \mathbb{F}_3^{13}.$$

Notations are as introduced in Construction 4.4 and Proposition 4.2.

Observe that $\mathcal{C}_0$ has minimum distance $d_0 = 3$. Note that $\boldsymbol{H}_0$ has the form

$$\boldsymbol{H}_0 = \begin{bmatrix} I_{d_0-2} & \boldsymbol{A} \\ \boldsymbol{0} & \boldsymbol{B} \end{bmatrix} = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 & 0 & 0 & 1 & 1 & 2 & 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 \end{pmatrix},$$

in Proposition 4.2 we can take $\boldsymbol{T}$ the identity matrix, and $\boldsymbol{Y} = \boldsymbol{H}_0$.

For a random message vector $\boldsymbol{m} = \begin{pmatrix} 1 & 1 & 1 & 2 & 1 & 1 & 0 \end{pmatrix} \in \mathbb{F}_3^7$,

$$\boldsymbol{w} = \boldsymbol{m}\boldsymbol{G}_1 = \begin{pmatrix} 1 & 1 & 1 & 2 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 2 \end{pmatrix} \in \mathbb{F}_3^{13}.$$

To mask the $d_0 - 2$ largest positions of the vector $\boldsymbol{s}$ (corresponding to the $(d_0 - 2)$ leftmost columns of $\boldsymbol{Y}$), namely s_0 (the leftmost value highlighted in blue in $\boldsymbol{s}$), the encoder sets $z_j = s_j - w_j$ for $j \in [d_0 - 2]$. Thus, $z_0 = 2 - 1 = 1$. Note that

$$\boldsymbol{w} + \begin{pmatrix} z_0 & 0 & 0 \end{pmatrix} \boldsymbol{H}_0 = \begin{pmatrix} 2 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}.$$

Next, the encoder determines z_1 and z_2 so as to mask in positions 5 and 6. As $\boldsymbol{H}_0$ has non-zero entries in the bottom row of columns 5 and 6, we have that $L_1 = \emptyset$ and $L_2 = \{5, 6\}$. The encoder can thus take any value for z_1, so let us say it takes 0. The coefficient z_2 is chosen in such a way that

$$\boldsymbol{w} + \begin{pmatrix} z_0 & 0 & 0 \end{pmatrix} \boldsymbol{H}_0 + \begin{pmatrix} 0 & 0 & z_2 \end{pmatrix} \boldsymbol{H}_0 \geq \boldsymbol{s}.$$

This inequality is satisfied if and only if $2 + z_2 \neq 0$ (position 5) and $0 + z_2 \neq 0$ (position

6), so if and only if $z_2 = 2$.

Note that, in this example, there are $\mathrm{wt}(\boldsymbol{s}) = 3$ *partially stuck cells. As* $\mathcal{C}$ *has minimum distance* 3*, it can correct a single error.* ▶

Corollary 4.2 (Generalization of Theorem 4.5)**.** *Let $s \in \mathbb{F}_q$ and let*

$$\Sigma = \left\{ \boldsymbol{s} \in \mathbb{F}_q^n \mid \mathrm{wt}(\boldsymbol{s}) \leq d_0 + \left\lceil \frac{q}{s} \right\rceil - 3 \text{ and } \max\{s_i \mid i \in [n] \leq s\} \right\}.$$

The coding scheme in Construction 4.4 is a $(\Sigma, \lfloor \frac{d-1}{2} \rfloor)$ *PSMC scheme of length n and size q^{k-l}.*

Proof. Let $\boldsymbol{s} \in \Sigma$ have weight $u \leq d_0 + \lceil \frac{q}{s} \rceil - 3$. Let $\Psi \subseteq [n]$ of size $n - d_0 + 2$ be such that the number of non-zero components of $\boldsymbol{s}$ in $[n] \setminus \Psi$ equals $\min(d_0 - 2, u)$. Then $\boldsymbol{s}$ has $u - \min(d_0 - 2, u)$ non-zero components in Ψ. As a consequence, if $u \leq d_0 - 2$, then $\sum_{i \in \Psi} s_i = 0$, and if $u > d_0 - 2$, then

$$\sum_{i \in \Psi} s_i \leq s(u - d_0 + 2) \leq s(\lceil \frac{q}{s} \rceil - 1) < s(\frac{q}{s} + 1 - 1) = q.$$

Hence in both cases, $\sum_{i \in \Psi} s_i \leq q - 1$. The corollary thus follows from Proposition 4.2. In particular, if $s = 1$, the corollary agrees with Theorem 4.5. □

4.6.3 Generalization of the Code Construction (up to $2^{\mu-1}(d_0 + 1) - 1$ Partially Stuck Cells)

We do not generalize Construction 4.5 as it is tailored to the *special* case where $s_i = 1$ for all $i \in \boldsymbol{\phi}$. Nevertheless, as the following remarks show, something is possible.

Remark 4.8. *A semi-generalization can be defined for $s_0 = s_1 = \cdots = s_{u-1}$ where $s_i > 1$ of the construction in Construction 4.5. We can have a construction similar to Construction 4.5, with alphabet size of the form $(s+1)^\mu$ and use an $(s+1)$-ary sub-field for masking. Any $(s+1)$-ary code is a subspace of codes over $(s+1)^\mu$ which analogs that a binary sub-field is subspace of our 2^μ-ary coding scheme. In this regard, the encoding process masks the remaining positions ($u_0 = \left\lfloor \frac{(s+1)u}{(s+1)^\mu} \right\rfloor$) of values strictly from the $(s+1)$ field, i.e., the base field. Thus, the following must hold: $\boldsymbol{H}_0 \in \mathbb{F}_{s+1}^{(l \times n)}$, $\boldsymbol{\gamma} \in \mathbb{F}_{s+1}^{l}$ in Step 2 in Algorithm 9, and the elements of the base field $(s+1)$ are the smallest ordered elements among the elements of $(s+1)^\mu$.*

Remark 4.9. *A generalization of Construction 4.5 to any arbitrary q levels, i.e., $\boldsymbol{s} \in \Sigma \subseteq \mathbb{F}_q^n$ can be conducted by the same considerations pursued by Proposition 4.2 but for q is strictly a power of the prime 2.*

4.7 Constructions for Unreachable Memory Cells (With Errors)

Section 3.4.4 introduces unreachable memory cells (UMC). In [WY16, Section VIII], the authors provide code constructions (without correcting additional errors) for unreachable memories at levels $\tilde{\boldsymbol{s}} := (q-1)^n - \boldsymbol{s}$, where $\boldsymbol{s} \in \Sigma \subset \mathbb{F}_q^n$ (see Definition 4.2). It is a dual problem of a PSMC scenario (cf. Section 3.4.3). In the sequel, we propose code constructions for UMC with correcting substitution errors, so we first describe our coding model concerning both.

Definition 4.4. *(Codes for Unreachable Memory Cells $(n, M)_q(\tilde{\Sigma}, t)$ UMC) For $\tilde{\Sigma} \subset \mathbb{F}_q^n$, $\tilde{\boldsymbol{s}} \in \tilde{\Sigma}$ and non-negative integer t, a q-ary $(\tilde{\Sigma},\, t)$-*unreachable-recovering code *$\mathcal{C}$ of length n and size M is a coding scheme consisting of a message set $\mathcal{M}$ of size M, an encoder $\mathcal{E}$ and a decoder $\mathcal{D}$ satisfying:*

1. *The encoder $\mathcal{E}$ is a mapping from $\mathcal{M} \times \tilde{\Sigma}$ to $\mathbb{F}_q^n$ such that*

$$\text{for each } (\boldsymbol{m}, \tilde{\boldsymbol{s}}) \in \mathcal{M} \times \tilde{\Sigma}, \quad \mathcal{E}(\boldsymbol{m}, \tilde{\boldsymbol{s}}) \leq \tilde{\boldsymbol{s}},$$

2. *For each $(\boldsymbol{m}, \tilde{\boldsymbol{s}}) \in \mathcal{M} \times \tilde{\Sigma}$ and each $\boldsymbol{e} \in \mathbb{F}_q^n$ such that*

$$\mathrm{wt}(\boldsymbol{e}) \leq t \text{ and } \mathcal{E}(\boldsymbol{m}, \tilde{\boldsymbol{s}}) + \boldsymbol{e} \leq \tilde{\boldsymbol{s}},$$

3. *It holds that*

$$\mathcal{D}(\mathcal{E}(\boldsymbol{m}, \tilde{\boldsymbol{s}}) + \boldsymbol{e}) = \boldsymbol{m}.$$

4.7.1 Equivalent Codes for Unreachable Memory Cells UMC

A code for unreachable cells maps message vectors on codewords that attain the unreachable cells, i.e., the values of each codeword at the unreachable cells coincide with their unreachable levels. Supposing similar encoding and decoding schemes as described in Figure 4.1 with respect to the UMC defective model (described in Definition 4.4), the encoder knows the locations and values of the unreachable cells, while the decoder does not. Hence, the task of the decoder is to reconstruct the message given only the codeword. The following construction makes a relation between PSMCs and UMCs.

Construction 4.6 (Constructions for UMC obtained by constructions for PSMC). *Let $\Sigma \subset \mathbb{F}_q^n$ such that*

$$\tilde{\Sigma} = \Big\{ \tilde{\boldsymbol{s}} \in \mathbb{F}_q^n \mid \exists \boldsymbol{s} \in \Sigma \,[\tilde{\boldsymbol{s}} = (q-1)^n - \boldsymbol{s}] \Big\}.$$

Then an $(n, M)_q(\tilde{\Sigma}, t)$ UMC can be constructed from an $(n, M)_q(\Sigma, t)$ PSMC.

Theorem 4.8 (Equivalent Codes for UMC)**.** *Let Construction 4.6 hold. A q-ary code $\mathcal{C}$ is an $(n, M)_q(\tilde{\Sigma}, t)$ UMC if and only if it is an $(n, M)_q(\Sigma, t)$ PSMC.*

Proof. Let there be $\Sigma \subset \mathbb{F}_q^n$ such that $\tilde{\Sigma} = \left\{ \tilde{\boldsymbol{s}} \in \mathbb{F}_q^n \mid \exists \boldsymbol{s} \in \Sigma \, [\tilde{\boldsymbol{s}} = (q-1)^n - \boldsymbol{s}] \right\}$. Wachter-Zeh and Yaakobi in [WY16, Theorem 15] for $t = 0$ prove that codes for PSMCs can be used to obtain codes for UMCs. Assuming $t = 0$ (i.e., no error correction is considered) and given $\boldsymbol{s}$ partially stuck levels, there is a $(\Sigma, 0)$ PSMC of redundancy $n-k$ by the constructions in Section 4.6.1 and Section 4.6.2. Thus, these constructions give $(\Sigma, 0)$ PSMCs that coincide with the constructions in [WY16] for $t = 0$. Hence, we infer that a $(\tilde{\Sigma}, t > 0)$ UMC with length n and size M can be constructed from a $(\Sigma, t > 0)$ PSMC with the same parameters. A (Σ, t) PSMC can correct at most t errors (cf. Theorem 2.3 in Section 2.3.3), so as $(\tilde{\Sigma}, t)$ UMC. The converse statement also holds, i.e., a (Σ, t) PSMC can be obtained from a $(\tilde{\Sigma}, t)$ UMC since $\boldsymbol{s} = (q-1)^n - \tilde{\boldsymbol{s}}$ by Construction 4.6. □

Remark 4.10. *The proof of Theorem 4.8 states that our constructions in Section 4.6.1 and Section 4.6.2 coincide with [WY16, Theorem 15] for t $= 0$ (error-free case in which masking only is needed). Thus, following the same argument in [WY16, page 651], our coding scheme for t > 0 offers an improvement from both [Hee83] and [GSD14] since $(\tilde{\Sigma}, t > 0)$ UMC with length n and size M can be constructed from a $(\Sigma, t > 0)$ PSMC with the same parameters.*

4.8 Errors Positions

In our code constructions, corruption can occur in any of n positions. If errors happen in the partially stuck cells' locations, they overlap with $\boldsymbol{\phi} \subseteq [n]$. Overlapping errors have been assumed to coincide with the partially stuck constraints. Construction 4.2 and Construction 4.3 disregard *one* prohibitive value (e.g., the *zero* subscripts) from the output vector in the partially stuck-at-1 positions. Overlapping errors, indeed, could violate this restriction in the previous constructions, e.g., $(c_i + e_i) \mod q = 0$ for $c_i = q-1$ and (coincidentally) $e_i = 1$ for $i \in \boldsymbol{\phi}$. Hence, our work in [AC22] deals with such a scenario to assure *zeros* cannot happen in $\boldsymbol{\phi}$ positions, given the intersecting vector $\boldsymbol{e} \in \{0, 1\}^n$. Therefore, we update the former constructions such that the encoded vector never attains the lowest and the highest levels, e.g., 0 and $q-1$ from the set $[q]$.

Accordingly, *two* values from the set $[q]$ in the output vector are forbidden. Thus, the modified schemes either cost more redundancy for masking while pondering violative[1] overlapping errors or handle fewer masked u cells while preserving our constructions redundancies, e.g., see the column "Redundancy" in Table 4.3 regarding Construction 4.2 and Construction 4.3.

[1]Overlapping errors can disobey the constraints of partially stuck cells.

In this dissertation, we skip these results for the sake of briefness as they are analogous to our earlier work in this chapter with a slight alteration in the encoding algorithms and a stricter condition on the error vector, e.g., $\boldsymbol{e} \in \{0,1\}^n$.

4.9 Open Problems and Observations

Coding for memories is a broad branch of knowledge that concerns many directions and code classes, e.g., [CZW08; MK09; ZPJ10; CKCH14; KH14; Wan$^+$14] to overcome *reliability* problems. Furthermore, some of the research paths, e.g., [Zha$^+$13], regards other measured factors like the *latency* (see Section 3.2.1) deploying low-density parity-check (LDPC) codes for this purpose.

Considering other metrics rather than the Hamming metric can be employed to provide coding models for partially stuck memory cells, e.g., *some special classes of rank metric codes* such as *interleaved codes* [WSS15; RPW21] and their properties can correct errors beyond the unique decoding radius (cf. Section 2.3.3). By using a collaborative approach, it is possible to correct a larger proportion of errors (beyond half of the minimum distance of the component code) in various algebraic *interleaved codes*, which can be viewed as *array codes* (cf. Appendix A.1) [Hol$^+$21]. Such coding schemes perfectly concur with the non-volatile memories thanks to their physical structure (see Section 3.2). Each cell is an array consisting of a few bits concurrently programmed to represent a distinct level. Errors then might be considered bit-wise that can be modeled as a matrix $\boldsymbol{E} \in \mathbb{F}_2^{\mu \times n}$. In this regard, a *burst* of errors with a certain *column weight* for an error matrix $\boldsymbol{E} \in \mathbb{F}_2^{\mu \times n}$, e.g., $\operatorname{supp}(\boldsymbol{E})$ (cf. Section 2.1), could be repaired.

We provide further directions, observations, and remarks about probable extensions of this thesis in Appendix A.1, Appendix A.2, and Appendix A.3. MDS codes (cf. Section 2.4.1) can be used to obtain polyalphabetic codes (cf. Section 2.5) over q_i for $i \in [n]$ such that each $[q_i] \subseteq \mathbb{F}_q$. Let $\mathcal{C}_{RS}$ be a Reed-Solomon (RS) codes [Rot06, Chapter 5] over $\mathbb{F}_q$ and $\mathcal{C}_{pol}$ be a polyalphabetic by Definition 2.9, then there is $\mathcal{C}_{pol} \subseteq \mathcal{C}_{RS}$. Hence, $\mathcal{C}_{RS}$ is called a *mother* code of $\mathcal{C}_{pol}$ [Sid$^+$05]. Remark A.1 in Appendix A.2 puts the *bedrock* to propose coding models based on polyalphabetic mother codes for partially stuck memory cells (cf. Definition 3.2) and unreachable memory cells (cf. Definition 3.3).

On the other hand, we exhibit short comparisons between codes covered for PSMC (defined in Section 3.4.3) with WOM codes (cf. Section 3.4.1) and rank modulation (cf. Section 3.4.2) through Remark A.2 and Remark A.3 in Appendix A.3. Note that a formal in-depth investigation could still be carried out in this regard. Combining two or more of the prior coding-built models perhaps brings more sophisticated coding schemes but might desirably present the optimum coding solutions for flash-like technologies.

5 Trading Partial Defects with Errors

Abstract

This chapter investigates a technique where the encoder, after a first masking step, introduces errors at some partially stuck positions of a codeword in order to satisfy the stuck-at constraints. The decoder uses part of the error-correcting capability to correct these introduced errors. It turns out that treating some partially stuck cells as erroneous cells can decrease the required redundancy for some code parameters, e.g., by Lemma 5.2.

The work in this chapter is based on our works in [APTW23b] that has been accepted in the 14th Annual Non-Volatile Memories Workshop (NVMW 2023) and in [APTW23a] that is in revision for publication in the journal Designs, Codes and Cryptography (DCC), 2023.

5.1 Introduction

RECALL that the encoder knows the faulty cells, while the decoder does not know any information about them. The task of an encoder in our code constructions in Chapter 4 is to recover from the partially stuck cells by coinciding with their values. It is possible to be directly consistent with the partially defective cells by the error-correcting code capability instead of sophisticated error-correcting and defect-masking schemes, e.g., Construction 4.2, Construction 4.4 and Construction 4.5. A linear block code $\mathcal{C}$ over $\mathbb{F}_q$ has the parameters, n, k, and $d \geq 2t+1$ (unique decoding, see Section 2.3.3) where $t = \mathrm{wt}(\boldsymbol{e})$ (cf. Section 2.3). The encoder can deliberately introduce errors in the partially stuck coordinates, i.e., setting them to 1; consequently, all locations are *masked* (non-zero). Now, suppose the code $\mathcal{C}$ has enough minimum distance d to correct the substitution errors and any other artificial errors (i.e., invented by the encoder). Then the decoder can correct these errors and successfully reconstruct the message. However, the larger the minimum distance suffices to correct both errors, the higher parity symbols are needed to define a code with that minimum distance.

5.1.1 Contributions and Outline

We start with Section 5.1.2 and provide Proposition 5.1 on how to correct the substitution errors and the artificial errors in the partially stuck-at positions. In the second part of this section, we show a general case through Theorem 5.1 that applies to all of our constructions (see Chapter 4), replacing any $0 \leq j \leq t$ errors with j masked partially stuck cells. Section 5.2 proposes improvements from Theorem 5.1, starting with Lemma 5.1. In the remainder of this section, we provide variations on the idea of the encoder introducing errors to the result of a first encoding step so that the final encoder output satisfies the partially stuck-at conditions. Lemma 5.2 and its generalized version Lemma 5.3 are for the prior purpose. The final part is Lemma 5.4, which considers Proposition 4.2 for arbitrary $\boldsymbol{s}$ levels to prove it is possible to mask partially stuck cells by the mean of the error-correcting code capability with improvement from Theorem 5.1.

5.1.2 General Theorem of Trading Partially Stuck Cells with Errors

In the constructions shown so far, the encoder output $\boldsymbol{c}$ is a word from an error correcting code $\mathcal{C}$. If $\boldsymbol{c}$ does not satisfy the partially-stuck-at conditions in j positions, the encoder could modify it in these j positions to obtain a word $\boldsymbol{c'} = \boldsymbol{c} + \boldsymbol{e'}$ satisfying the partially-stuck-at constrains, while $\mathrm{wt}(\boldsymbol{e'}) = j$. If $\mathcal{C}$ can correct t errors (cf. Theorem 2.3 in Section 2.3.3), then it still is possible to correct $t - j$ errors in $\boldsymbol{c'}$. This observation was also made in [Hee83, Theorem 1]. The above reasoning shows that the following proposition holds.

Proposition 5.1. *If there is an $(n, M)_q(u, 1, t)$ PSMC, then for any j with $0 \leq j \leq t$, there is an $(n, M)_q(u + j, 1, t - j)$ PSMC.*

We generalize the above proposition to general Σ (Theorem 5.1).

Theorem 5.1 (Partial Masking PSMC)**.** *Let $\Sigma \subset \mathbb{F}_q^n$, and assume that there exists an $(n, M)_q(\Sigma, t)$ PSMC $\mathcal{C}$. For any $j \in [t]$, there exists an $(n, M)_q(\Sigma^{(j)}, t - j)$ PSMC $\mathcal{C}_j$, where*

$$\Sigma^{(j)} = \Big\{ \boldsymbol{s'} \in \mathbb{F}_q^n \mid \exists \boldsymbol{s} \in \Sigma \, [d(\boldsymbol{s}, \boldsymbol{s'}) \leq j \text{ and } \boldsymbol{s'} \geq \boldsymbol{s}] \Big\}.$$

Proof. Let the encoder $\mathcal{E}_j$ and the decoder $\mathcal{D}_j$ for $\mathcal{C}_j$ be Algorithm 11 and Algorithm 12, respectively. By definition, $\boldsymbol{c'} \geq \boldsymbol{s'}$. Moreover, if $s_i = s'_i$, then $c_i \geq s_i = s'_i$, so $c_i = c'_i$. As a result, $d(\boldsymbol{c}, \boldsymbol{c'}) \leq j$.

In Algorithm 12, the decoder $\mathcal{D}$ of $\mathcal{C}$ is directly used for decoding $\mathcal{C}_j$. As $\boldsymbol{y} \geq \boldsymbol{s'}$, surely $\boldsymbol{y} \geq \boldsymbol{s}$. Moreover, we can write $\boldsymbol{y} = \boldsymbol{c} + (\boldsymbol{c'} - \boldsymbol{c} + \boldsymbol{e})$. As shown above, $\mathrm{wt}(\boldsymbol{c'} - \boldsymbol{c}) \leq j$, and so $\mathrm{wt}(\boldsymbol{c} - \boldsymbol{c'} + \boldsymbol{e}) \leq t$. As a consequence, $\mathcal{D}(\boldsymbol{y}) = \boldsymbol{m}$. □

We can improve on Theorem 5.1 for Construction 4.5 giving Lemma 5.1.

Algorithm 11: Encoding

```
  Input: $(\boldsymbol{m}, \boldsymbol{s}') \in \mathcal{M} \times \Sigma^{(j)}$.
1 Determine $\boldsymbol{s} \in \Sigma$ such that $d(\boldsymbol{s}, \boldsymbol{s}') \leq j$ and $\boldsymbol{s}' \geq \boldsymbol{s}$.
2 Let $\boldsymbol{c} = \mathcal{E}(\boldsymbol{m}, \boldsymbol{s})$.
3 Define $\boldsymbol{c}' = \mathcal{E}'_j(\boldsymbol{m}, \boldsymbol{s}')$ as $c'_i = \max(c_i, s'_i)$ for $i \in [n]$.
  Output: Codeword $\boldsymbol{c}'$.
```

Algorithm 12: Decoding

```
  Input: Received $\boldsymbol{y} = \boldsymbol{c}' + \boldsymbol{e}$ where $\mathrm{wt}(\boldsymbol{e}) \leq t - j$ and $\boldsymbol{y} \geq \boldsymbol{s}'$
1 Message $\boldsymbol{m} = \mathcal{D}(\boldsymbol{y})$
  Output: Message vector $\boldsymbol{m}$
```

5.2 Improvements of the General Theorem

The following lemmas enhance Theorem 5.1.

5.2.1 Based on our Binary-established Construction

The following lemma corresponds to Construction 4.5.

Lemma 5.1. *Given an $[n, k, d]_q$ code as defined in Construction 4.5, then for any j such that $0 \leq j \leq \lfloor \frac{d-1}{2} \rfloor$, there is a 2^{μ}-ary $(2^{\mu-1}(d_0 + j) - 1, 1, \lfloor \frac{d-1}{2} \rfloor - j)$ PSMC of length n and size q^{k-l-1}.*

Proof. Let $\boldsymbol{\phi} \subset [n]$ has size $u \leq 2^{\mu-1}(d_0+j)-1$. We use the notation from Algorithm 9. After Step 1, $\boldsymbol{w}$ has at most $u_0 = \lfloor \frac{2u}{2^{\mu}} \rfloor \leq d_0 + j - 1$ binary entries in the positions from $\boldsymbol{\phi}$. After Step 2, at least $d_0 - 1$ of these entries in $\boldsymbol{c}$ differ from 0. By setting the at most j other binary entries in the positions from $\boldsymbol{\phi}$ equal to 1, the encoder introduces at most j errors, and guarantees that the partially-stuck-at conditions are satisfied. □

5.2.2 Based on Another Approach for Introducing Errors

The following lemma is for Construction 4.2 as follows. In this lemma, we use another approach for introducing errors in order to satisfy the stuck-at conditions.

Lemma 5.2. *Given an $[n, k, d]_q$ code containing a word of weight n, for any j with $0 \leq j \leq \lfloor \frac{d-1}{2} \rfloor$, there is a q-ary $(q - 1 + qj, 1, \lfloor \frac{d-1}{2} \rfloor - j)$ PSMC of length n and size q^{k-1}.*

Proof. We use the notation from Construction 4.2.

Let $\boldsymbol{\phi} \subset [n]$ have size $u \leq q-1+qj$. Let $\boldsymbol{x}$ be a codeword of weight n. For each $i \in \boldsymbol{\phi}$, there is exactly one $v \in \mathbb{F}_q$ such that $w_i + vx_i = 0$, and so

$$\sum_{v \in \mathbb{F}_q} |\{i \in \boldsymbol{\phi} \mid w_i + vx_i = 0\}| = u.$$

As a consequence, there is $v \in \mathbb{F}_q$ such that $\boldsymbol{c} = \boldsymbol{w} + v\boldsymbol{x}$ has at most $\lfloor \frac{u}{q} \rfloor \leq j$ entries in $\boldsymbol{\phi}$ equal to zero. By setting these entries of $\boldsymbol{c}$ to a non-zero value, the encoder introduces at most j errors. As $\mathcal{C}$ can correct up to $\lfloor \frac{d-1}{2} \rfloor$ errors, it can correct these j errors and additionally up to $\lfloor \frac{d-1}{2} \rfloor - j$ substitution errors. □

Example 5.1. *Consider a $[15, 9, 5]_4$ code $\mathcal{C}$ containing the all-one word, e.g. the BCH code with zeroes $\alpha, \alpha^2, \alpha^3$ (see Section 2.3.6), where α is a primitive element in $\mathbb{F}_{16}$ (see Definition 2.3). Let $u \leq 7$ and $t = 1$. We use the all-one word for partial masking, ensuring that 0 occurs in at most $\lfloor \frac{u}{4} \rfloor \leq 1$ position indexed by $\boldsymbol{\phi}$. We set the codeword value in this position to 1, introducing one error. We can correct this introduced error and one additional random error as $\mathcal{C}$ has minimum distance 5. Hence, we have obtained a 4-ary $(7, 1, 1)$ PSMC of length 15 and cardinality 4^8.* ▶

We show in Example 5.2 how applying Lemma 5.2 for Construction 4.5 outperforms Lemma 5.1.

Example 5.2. *Given $d_0 = 3$, $u = 15$ and $q = 2^2$ and let α be a primitive element in $\mathbb{F}_4$ (see Definition 2.3) and take $\boldsymbol{x} = \mathbf{1}$. Assume we have*

$$\begin{aligned} \boldsymbol{w}^{(\phi)} &= (\boldsymbol{m}' \cdot \boldsymbol{H}_0 + \boldsymbol{m} \cdot \boldsymbol{G}_1) + z \cdot \mathbf{1} \\ &= (0, 1, \alpha, 1+\alpha, 0, 1, \alpha, 1+\alpha, 0, 1, \alpha, 1+\alpha, 0, 1, \alpha) \\ &+ z \cdot \mathbf{1}, \end{aligned}$$

then choosing $z = 1 + \alpha$ minimizes the number of binary values in $\boldsymbol{w}^{(\phi)}$, we get:

$$\boldsymbol{w}^{(\phi)} = (1+\alpha, \alpha, 1, 0, 1+\alpha, \alpha, 1, 0, 1+\alpha, \alpha, 1, 0, 1+\alpha, \alpha, 1).$$

Following Step 2 in Algorithm 9 and since $d_0 = 3$, we can mask at most $d_0 - 1$ binary values highlighted in the vector $\boldsymbol{w}^{(\phi)}$ that leaves us, in this example, with at most $\lfloor \frac{2u}{2^2} \rfloor - d_0 + 1 = 5$ zeros that remain unmasked.
However, applying Lemma 5.2 instead for Construction 4.5 gives a better result. Choosing $\boldsymbol{\gamma} = \mathbf{0}$ in Step 2 of Algorithm 9, we obtain $\boldsymbol{c}^{(\phi)} = \boldsymbol{w}^{(\phi)}$ with $(\lfloor \frac{u}{q} \rfloor = 3)$ zeros highlighted in blue above that we can directly trade. ▶

Remark 5.1. *As the code from Construction 4.5 has a word of weight n, Lemma 5.2 implies the existence of an $(u, 1, t)$ PSMC of cardinality q^{k-1} under the condition that $2(t + \lfloor \frac{u}{q} \rfloor) < d$. Lemma 5.1 shows the existence of an $(u, 1, t)$ PSMC of smaller cardinality,* viz. *q^{k-l}, under the condition that $2(t + \max(0, \lfloor \frac{2u}{2^\mu} \rfloor - d_0 + 1) < d$. As a consequence, Lemma 5.1 can only improve on Lemma 5.2 if $d_0 - 1 > \lfloor \frac{2u}{2^\mu} \rfloor - \lfloor \frac{u}{2^\mu} \rfloor$.*

5.2.3 Based on Another Approach for Introducing Errors Considering Arbitrary Partially Stuck Levels

We can generalize Lemma 5.2 as follows.

Lemma 5.3. *Given an $[n, k, d]_q$ code containing a word of weight n. Let $0 \leq j \leq \lfloor \frac{d-1}{2} \rfloor$, and let*

$$\Sigma = \left\{ \boldsymbol{s} \in \mathbb{F}_q^n \;\middle|\; \sum_i s_i \leq q - 1 + qj \right\}.$$

There is a q-ary $(\Sigma, \lfloor \frac{d-1}{2} \rfloor - j)$ PSMC of length n and size q^{k-1}.

Proof. We use the notation from Theorem 4.7. For simplicity, we assume that the code contains the all-one word. We wish to choose the multiplier $v \in \mathbb{F}_q$ such that $\boldsymbol{c} = \boldsymbol{w} - v \cdot \mathbf{1}$ satisfies $c_i \geq s_i$ for as many indices i as possible. For each index i, there are $q - s_i$ values for v such that this inequality is met. Hence, there is a $v \in \mathbb{F}_q$ such that $c_i \geq s_i$ for at least $\lceil \frac{1}{q} \sum_i (q - s_i) \rceil = n - \lfloor \frac{1}{q} \sum_i s_i \rfloor$ indices i. The encoder thus needs to introduce errors only in the at most $\lfloor \frac{1}{q} \sum_i s_i \rfloor$ positions for which the inequality is not satisfied. □

5.2.4 Based on our Generalized Construction for Arbitrary Partially Stuck Levels

We next show via Lemma 5.4 that trading partially stuck cells (for any arbitrary stuck levels $\boldsymbol{s}$, see Proposition 4.2) with errors works while improving upon Theorem 5.1.

Lemma 5.4. *Assume there exists a matrix as in Proposition 4.2. Let $0 \leq j \leq \lfloor \frac{d-1}{2} \rfloor$, and let*

$$\Sigma = \left\{ \boldsymbol{s} \in \mathbb{F}_q^n \;\middle|\; \exists \Psi \subset [n] : \;\middle|\; \Psi \;\middle|\; = n - d_0 + 2 \left[\sum_{i \in \Psi} s_i \leq q - 1 + qj \right] \right\}.$$

Then exists a q-ary $(\Sigma, \lfloor \frac{d-1}{2} \rfloor - j)$ PSMC of length n and size q^{k-l}.

Proof. Let $\boldsymbol{s} \in \Sigma$. In order to simplify notation, we assume without loss of generality that $\sum_{i=d_0-2}^{n-1} s_i \leq q - 1 + qj$. We use the same argument as in the alternative proof of Proposition 4.2 (see Appendix A.4). Clearly,

$$n - d_0 + 2 - \left\lfloor \frac{1}{q} \sum_{i=d_0-2}^{n-1} s_i \right\rfloor \geq n - d_0 + 2 - j.$$

So we infer that for at least $n - j$ indices $i \in [n]$,

$$w_i + \Big((\boldsymbol{z}, \boldsymbol{\eta}) \boldsymbol{T} \boldsymbol{H}_0 \Big)_i \geq s_i.$$ □

Remark 5.2. *The proof of Lemma 5.4 shows that the encoder output in fact can be made equal to $\boldsymbol{s}$ in the $d_0 - 2$ largest entries of $\boldsymbol{s}$. In fact, it shows that the scheme allows for masking $d_0 - 2$ stuck-at errors, masking partial stuck errors in the remaining cells, and correcting $\lfloor \frac{d-1}{2} \rfloor - j$ substitution errors, provided that the sum of the partially stuck-at levels in the $n - d_0 + 2$ remaining cells is less than $(j+1)q$.*

Notice that by using Lemma 5.2 and Lemma 5.3, we gain (for example, for $j = 1$) exactly $2^{\mu-1}$ and 2^{μ} more masked partially stuck cells under the condition that $2(t + \lfloor \frac{u}{q} \rfloor) < d$. To the present, the only way to determine whether or not it is profitable to introduce errors into partially stuck cells is numerical. These results that compare applying Theorem 5.1, Lemma 5.1, and Lemma 5.2 are found in Section 6.3.4 and Section 6.3.4 of Chapter 6. Lemma 5.2 (see Figure 6.8) suggests that treating some partially stuck cells as erroneous cells can reduce the necessary redundancy for some code parameters.

6
Bounds on Memories with Defects and Errors

Abstract

We derive Singleton-like, sphere-packing-like, and Gilbert–Varshamov-like bounds regarding our code constructions in Chapter 4. We numerically compare our derived upper- and lower-type bounds. Our sphere-packing-like bound has been compared to the usual sphere-packing upper bound, and for the case of no errors ($t = 0$) to [WY16, Theorem 2]. Our lower-type bounds, for given (u, t), are compared to each other and to classical Gilbert–Varshamov bound for $(q-1)$-ary codes. The exchange of a one error correction ability with a single masking capability of a partially stuck cell has also been demonstrated in these comparisons. Numerical comparisons state that our constructions match the Gilbert–Varshamov-like bounds for several code parameters, e.g., BCH codes that contain all-one word by Construction 4.2.

Some work in this chapter is based on our works in [APW20] that has been published in the 2020 Algebraic and Combinatorial Coding Theory (ACCT), in [APW21] that has been accepted in the 12th Annual Non-Volatile Memories Workshop (NVMW 2021), in [APTW23b] that has been accepted in the 14th Annual Non-Volatile Memories Workshop (NVMW 2023), and in [APTW23a] that is in revision for publication in the journal Designs, Codes and Cryptography (DCC), 2023.

6.1 Introduction

THE upper limits (see Sections 2.4.1, 2.4.2, 2.4.3 and 2.4.4) are used to evaluate the parameters of a code as necessary conditions. On the other hand, the lower bound (*Gilbert-Varshamov* bound, cf. Section 2.4.5) provides a sufficient condition on the existence of a code for given parameters, n, k, d, and q. We have provided various constructions in Chapter 4 based on q-ary t-error correcting codes with further prerequisites. We want to derive upper and lower limits on the size of these constructions.

6.1.1 Contributions and Outline

In this chapter, we derive bounds on our code constructions for PSMCs (from Chapter 4), namely Singleton-type, sphere-packing-type, and Gilbert-Varshamov-type bounds. The output of an encoder can achieve all the elements from $\mathbb{F}_q$; in the other cells where stuck-at is supposed, it confines the values in the partially stuck-at cells. So the set of all encoder outputs constructs a poly-alphabetic code (see Section 2.5), i.e., taking into account that all the allowed values in the partially stuck cells are sub-alphabets from the set $\mathbb{F}_q$. Hence, we show in Section 6.2 that upper bounds, namely *Singleton-type* and *sphere-packing-type*, on the size of poly-alphabetic codes (see Section 2.5.1) are also upper bounds on the size of partially stuck-at codes. Our sphere-packing-like bound for the size of (Σ, t) PSMCs has been compared to the standard sphere-packing upper bound (cf. Section 2.4.4) and for the case of no errors ($t = 0$) to [WY16, Theorem 2], located in Section 6.2.3.

Sufficient conditions for the existence of matrices satisfying the conditions from our constructions in Chapter 4 have been extensively investigated. We employ *Gilbert–Varshamov-like* techniques to show the existence of $(u, 1, t)$ PSMCs in a *finite* regime. In the finite GV consideration, Section 6.3.1 (see also Appendix A.5) presents these outcomes. A $(q-1)$-ary code of length n with a minimum distance of at least $2t+1$ is a q-ary $(u, 1, t)$ PSMC of length n with $u = n$, i.e., such a code can avoid the 0 subscripts from the set $[q]$ in the partially stuck positions. Thus, we give a corollary to combine this observation with the Gilbert bound for a $(q-1)$-ary alphabet in the final part of Section 6.3.1.

We have compared our GV-type bounds, for given (u, t), to each other and to $(q-1)$-ary codes. Our coding method in Section 4.5.1 for the case $u < q$ is to have a generator matrix consisting of the all-one row vector, i.e., a code containing the all-one word. To verify our findings related to the aforementioned case, we numerically compare, located in Section 6.3.2, to other upper boundaries like Griesmer and Ball–Blokhuis bounds, defined in Sections 2.4.2 and 2.4.3, respectively. On top of that, we compare to BCH codes (see Section 2.3.6) that possess all-one codewords.

In the same section, we also provide different comparisons between our code constructions and the existence of the code based on Theorem 6.1, Theorem 6.2 and Theorem 6.3. Then, we discuss and compare that it can be advantageous to introduce errors in some partially stuck cells in order to satisfy the stuck-at constraints as described in Section 5.1.2. We further extend our results to cover the existence of $(u, 1, t)$ PSMCs in the *asymptotic* version of the GV bound, given in Section 6.3.3. In the asymptotic model of our GV-like bounds, partial analytical comparisons have been discussed in Section 6.3.3, and also confirmed numerically, e.g., via Figure 6.6.

The last part, Section 6.4, briefly summarizes our lower/upper bounds on polyalphabetic codes when *alphabets sizes are bounded*, i.e., unlike [Sid$^+$05] that considers arbitrary alphabet sizes (cf. Section 2.5). Our outcomes confirm that the Elias-Bassalygo bound [Bas65; Joh63] and the first linear-programming bound [Lev98; MRRW77] are *tighter* for codes with some minimum distances for *finite* alphabet sizes.

6.2 Upper Bounds on Codes for Partially Stuck Memory Cells (PSMCs)

The output of an encoder has restrictions on the values in the partially-stuck-at cells; in the other cells, it can attain all values. So the set of all encoder outputs is a poly-alphabetic code [Sid+05], reformulated in Section 2.5. To be more precise, the following proposition holds.

6.2.1 Singleton-type Bound on PSMCs

Proposition 6.1. *Let $\mathcal{C}$ be an $(n, M)_q(\Sigma, t)$ partially-stuck-at-masking code with encoder $\mathcal{E}$. For any $\boldsymbol{s} \in \Sigma$, let*

$$\mathcal{C}_{\boldsymbol{s}} = \{\mathcal{E}(\boldsymbol{m}, \boldsymbol{s}) \mid \boldsymbol{m} \in \mathcal{M}\}.$$

Then $\mathcal{C}_{\boldsymbol{s}}$ is a code with minimum distance at least $2t+1$ and $|\mathcal{M}|$ words, and

$$\mathcal{C}_{\boldsymbol{s}} \subset Q_0 \times Q_1 \times \cdots \times Q_{n-1}, \text{ where}$$

$Q_i = \{x \in \mathbb{F}_q \mid x \geq s_i\}$.

Proof. By our error model, errors in stuck-at cells result in values still satisfying the stuck-at constraints. Therefore, t errors can be corrected (see Theorem 2.3 in Section 2.3.3) if and only if $\mathcal{C}_{\boldsymbol{s}}$ has minimum Hamming distance at least $2t+1$. The rest of the proposition is obvious. □

As a result of Proposition 6.1, upper bounds on the size of poly-alphabetic codes [Sid+05] are also upper bounds on the size of partially-stuck-at codes. Hence, we give the following corollaries to state Singleton-type and sphere-packing-type bounds for error-correcting and partially-stuck-at-masking codes.

Corollary 6.1. *(Singleton-type bound) Let $\mathcal{C}$ be a q-ary (Σ, t) PSMC of length n and size M. Then for any $\boldsymbol{s} = (s_0, \ldots, s_{n-1}) \in \Sigma$,*

$$M \leq \min\left\{ \prod_{j\in J}(q - s_j) \;\middle|\; J \subset [n], |J| = n - 2t \right\}.$$

Proof. Combination of Proposition 6.1 and Theorem 2.9. □

6.2.2 Sphere-packing-type Bound on PSMCs

Corollary 6.2. *(Sphere-packing type bound) Let $\mathcal{C}$ be a q-ary (Σ, t) PSMC of length n and size M. Then for any $\boldsymbol{s} = (s_0, \dots, s_{n-1}) \in \Sigma$*

$$M \leq \frac{\prod_{i=0}^{n-1}(q - s_i)}{V_t^{(b)}},$$

where $V_t^{(b)}$, the volume of a ball of radius t, satisfies

$$V_t^{(b)} = \sum_{r=0}^{t} V_r^{(s)},$$

where the volume $V_r^{(s)}$ of the sphere with radius r is given by

$$V_0^{(s)} = 1,$$

$$V_r^{(s)} = \sum_{1 \leq i_1 < \dots < i_r \leq n} (q - 1 - s_{i_1}) \cdots (q - 1 - s_{i_r}).$$

Proof. Combination of Proposition 6.1 and Theorem 2.10. □

Remark 6.1. *The difference between poly-alphabetic codes and partially-stuck-at-masking codes is that in the former, the positions of stuck-at cells and the corresponding levels are known to both encoder and decoder, whereas in the latter, this information is known to the encoder only.*

6.2.3 Discussion and Numerical Results

Figure 6.1 compares our derived sphere-packing-like bound to the amount of storable information symbols for a completely reliable memory (i.e., no stuck cells, no errors that can be seen at $u = 0$ in the solid line) and the upper bound on the cardinality of an only-masking PSMC (only stuck cells, no errors) derived in [WY16] as depicted in the solid curve. At $u = 0$, the derived sphere-packing-type bound (dotted and dashed-dotted plots) matches the usual sphere-packing bound (defined in Section 2.4.4) ("only errors") case. The more u partially-stuck-at cells, the less amount of storable information, i.e., only $q - 1$ levels can be utilized. Hence, the dotted and dashed-dotted lines are declining while u is growing. On the other hand, the more errors (e.g., $t = 25$ in the dashed-dotted plot), the higher overall required redundancy and the lower storable information for the aforementioned curve.

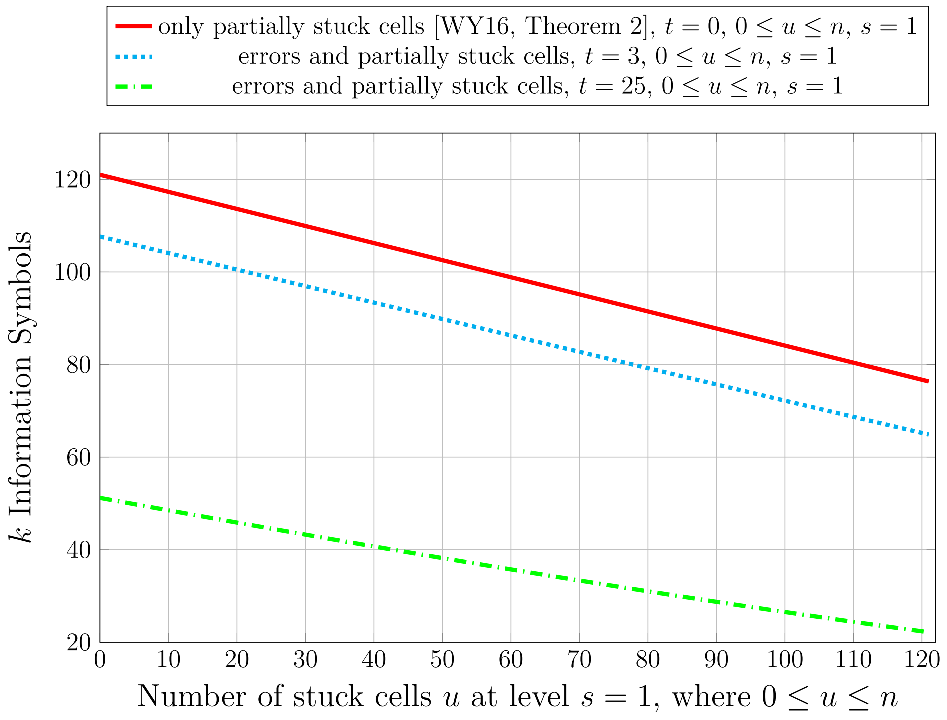


Figure 6.1: **Sphere-packing bounds**: Comparison for k information symbols for ("only partially stuck cells [WY16, Theorem 2]") and our sphere-packing-like ("errors and partially stuck cells") bounds. The classical sphere-packing bound ("only errors") can read at $u = 0$ in our sphere-packing-like bounds curves. The chosen parameters are $\mu = 5$ and $q = 3$, and $n = ((q^{\mu} - 1)/(q - 1))$.

6.3 Lower Bound on Codes for Partially Stuck Memory Cells (PSMCs)

In the following sections, we show sufficient conditions for the existence of matrices that meet the requirements from our coding models in Chapter 4.

6.3.1 Finite Gilbert–Varshamov-type (GV-type) Bound on PSMCs

We utilize *Gilbert–Varshamov-like* strategies (cf. Theorem 2.7) to establish the presence of $(u, 1, t)$ PSMCs at first in a *finite* regime (i.e., the code length n does not tend to infinity). Next in Section 6.3.3, we study the asymptotic (cf. to Theorem 2.8) of the resulting GV-type bounds.

Gilbert-Varshamov-like bound by Construction 4.2

Construction 4.2 requires a code that includes the all-one codeword. Therefore, we show the existence of a code that contains the all-one vector as a codeword. In the *subsequent section*, we will show that Gilbert-Varshamov-like bound from Theorem 4.2 is a special case of Theorem 6.1, given below. Furthermore, we provide earlier findings for Gilbert-Varshamov-like bound from Theorem 4.2 that were quite interesting and published in [APW20]. We restate the latter in Appendix A.5.

We start with a somewhat refined version of the Gilbert bound, that should be well-known, but for which we did not find an explicit reference.

Lemma 6.1. *Let q be a prime power, and assume there is an $[n, s]_q$ code $\mathcal{C}_s$ with minimum distance at least d. If $k \geq s$ is such that*

$$\sum_{i=0}^{d-1} \binom{n}{i} (q-1)^i < q^{n-k+1},$$

then there is an $[n, k]_q$ code $\mathcal{C}_k$ with minimum distance at least d that has $\mathcal{C}_s$ as a subcode.

Proof. By induction on k. For $k = s$, the statement is obvious. Now let $\kappa \geq s$ and let $\mathcal{C}_\kappa$ be an $[n, \kappa]_q$ code with minimum distance at least d that has $\mathcal{C}_s$ as a subcode. If $q^\kappa \sum_{i=0}^{d-1} \binom{n}{i}(q-1)^i < q^n$, then the balls with radius $d-1$ centered at the words of $\mathcal{C}_\kappa$ do not cover $\mathbb{F}_q^n$, so there is a word $\boldsymbol{x}$ at distance at least d from all words in $\mathcal{C}_\kappa$. As shown in the proof of [Lin92, Theorem. 5.1.8], the $[n, \kappa+1]_q$ code $\mathcal{C}_{\kappa+1}$ spanned by $\mathcal{C}_\kappa$ and $\boldsymbol{x}$ has minimum distance at least d. □

Finite GV bound based on Lemma 5.2

The following theorem is based on Lemma 5.2 that also proves (see Remark 6.2 next) the existence of $(u, 1, t)$ PSMCs from Construction 4.2 (cf. Section 4.5.1).

Theorem 6.1. *Let q be a prime power. Let n, k, t, u be non-negative integers such that*

$$\sum_{i=0}^{2(t+\lfloor \frac{u}{q} \rfloor)} \binom{n}{i} (q-1)^i < q^{n-k+1}.$$

There exists a q-ary $(u, 1, t)$ PSMC of length n and size q^{k-1}.

Proof. Let $\mathcal{C}_1$ be the $[n, 1, n]_q$ code generated by the all-one word. Lemma 6.1 implies that there is an $[n, k]_q$ code with minimum distance at least $2(t + \lfloor \frac{u}{q} \rfloor) + 1$ that contains the all-one word. Lemma 5.2 shows that $\mathcal{C}_k$ can be used to construct a PSMC with the claimed parameters. □

Remark 6.2. *GV bound from Theorem 4.2 is a special case of Theorem 6.1 for $u \leq q-1$.*

Gilbert-Varshamov-like bound by Construction 4.4

Regarding Construction 4.4 (in Section 4.5.2), we begin with the following lemmas that provide the bedrock to Theorem 6.2 to demonstrate the presence of $(u, 1, t)$ PSMCs.

Lemma 6.2. *Let q be a prime power, and let $1 \leq k < n$. Let $E \subset \mathbb{F}_q^n \setminus \{0\}$. The fraction of $[n,k]_q$ codes with non-empty intersection with E is less than $|E|/q^{n-k}$.*

Proof. Let $\mathcal{C}$ be the set of all $[n,k]_q$ codes. Obviously,

$$\left|\left\{C \in \mathcal{C} \mid C \cap E \neq \emptyset\right\}\right| \leq \sum_{C \in \mathcal{C} : C \cap E \neq \emptyset} \left| C \cap E \right| = \sum_{C \in \mathcal{C}} \left| C \cap E \right|.$$

It follows from [Loe97, Lemma 3] that

$$\frac{1}{|\, \mathcal{C} \,|} \sum_{C \in \mathcal{C}} \left| C \cap E \right| = \frac{q^k - 1}{q^n - 1} |E| < \frac{|E|}{q^{n-k}}. \qquad \square$$

Remark 6.3. *If E has the additional property that $\lambda \boldsymbol{e} \in E$ for all $\boldsymbol{e} \in E$ and $\lambda \in \mathbb{F}_q \setminus \{0\}$, then the upper bound in Lemma 6.2 can be reduced to $|E|/(q-1)q^{n-k}$.*

Lemma 6.3. *Let k, n, d and $d^{\perp}$ be integers such that*

$$\sum_{i=0}^{d-1} \binom{n}{i} (q-1)^i < \frac{1}{2} q^{n-k} \text{ and}$$
$$\sum_{i=0}^{d^{\perp}-1} \binom{n}{i} (q-1)^i < \frac{1}{2} q^{k}.$$

There exists a q-ary $[n,k]$ code $\mathcal{C}$ with minimum distance at least d such that $\mathcal{C}^{\perp}$ has minimum distance at least $d^{\perp}$.

Proof. Let $\mathcal{C}$ denote the set of all $[n,k]_q$ codes. By applying Lemma 6.2 with $E = \{\boldsymbol{e} \in \mathbb{F}_q^n \mid 1 \leq \mathrm{wt}(\boldsymbol{e}) \leq d-1\}$ and using the first condition of the lemma, we see that more than half of the codes in $\mathcal{C}$ have empty intersection with E, that is, have minimum distance at least d. Similarly, more than half of all q-ary $[n, n-k]$ codes have minimum distance at least $d^{\perp}$, and so more than half of the codes in $\mathcal{C}$ have a dual with minimum distance at least $d^{\perp}$. We conclude that $\mathcal{C}$ contains a code with both desired properties. $\square$

Theorem 6.2 (Gilbert-Varshamov-like bound by Construction 4.4)**.** *Let q be a prime power. Suppose the positive integers u, t, n, k, l with $u, t \leq n$ and $l < k$ satisfy*

$$\sum_{i=0}^{2t} \binom{n}{i} (q-1)^i < \frac{1}{2} q^{n-l}, \tag{6.1}$$

$$\sum_{i=0}^{u-q+2} \binom{n}{i}(q-1)^i < \frac{1}{2}q^l, \tag{6.2}$$

$$\sum_{i=0}^{2t} \binom{n}{i}(q-1)^i < q^{n-k+1}. \tag{6.3}$$

Then there is a q-ary $(u, 1, t)$ PSMC of length n and cardinality q^{k-l}.

Proof. According to Lemma 6.3, (6.1) and (6.2) imply the existence of an $[n, l]_q$ code $\mathcal{C}_0$ with minimum distance at least $2t + 1$ for which the dual code has minimum distance at least $u - q + 3$. Lemma 6.1 shows that $\mathcal{C}_0$ can be extended to an $[n, k]_q$ code $\mathcal{C}$ with minimum distance at least d. As $\mathcal{C}$ has a generator matrix of the form required by Construction 4.4, the theorem follows. □

Gilbert-Varshamov-like bound by Construction 4.5

In this section, we give sufficient conditions for the existence of matrices satisfying the conditions of Proposition 4.1 (located in Section 4.5.2) since it achieves larger sizes of coding schemes produced from a modified version of Construction 4.5.

We start with Lemmas 6.4 and 6.5, then we prove the main theorem (Theorem 6.3).

Lemma 6.4. *Let $\boldsymbol{G}$ be a $k \times n$ matrix over $\mathbb{F}_q$. For $s \geq 1$, let*

$$d_s = \min\{\mathrm{wt}(\boldsymbol{mG}) \mid \boldsymbol{m} \in \mathbb{F}_{q^s}^k \setminus \{\boldsymbol{0}\}\}.$$

Then $d_s = d_1$.

Proof. Let $s \geq 1$. As $\mathbb{F}_q \subseteq \mathbb{F}_{q^s}$, it is clear that $d_1 \geq d_s$.

To show the converse, we use the trace function T defined as $T(x) = \sum_{i=0}^{s-1} x^{q^i}$. As is well-known, T is a non-trivial mapping from $\mathbb{F}_{q^s}$ to $\mathbb{F}_q$, and

$$T(ax + by) = aT(x) + bT(y), \tag{6.4}$$

for all $x, y \in \mathbb{F}_{q^s}$ and $a, b \in \mathbb{F}_q$. We extend the trace function to vectors by applying it coordinate-wise.

Let $\boldsymbol{m} \in \mathbb{F}_{q^s}^k \setminus \{\boldsymbol{0}\}$. We choose $\lambda \in \mathbb{F}_{q^s}$ such that $T(\lambda \cdot \boldsymbol{m}) \neq \boldsymbol{0}$. As $T(0) = 0$, we infer that $\mathrm{wt}(\boldsymbol{mG}) = \mathrm{wt}(\lambda \cdot \boldsymbol{mG}) \geq \mathrm{wt}(T(\lambda \cdot \boldsymbol{mG}))$. As all entries from $\boldsymbol{G}$ are in $\mathbb{F}_q$, it follows from (6.4) that $T(\lambda \cdot \boldsymbol{mG}) = T(\lambda \cdot \boldsymbol{m})\boldsymbol{G}$. As a consequence,

$$\mathrm{wt}(\boldsymbol{mG}) \geq \mathrm{wt}(T(\lambda \cdot \boldsymbol{mG})) = \mathrm{wt}(T(\lambda \cdot \boldsymbol{m})\boldsymbol{G}) \geq d_1,$$

where the last inequality holds as $T(\lambda \cdot \boldsymbol{m}) \in \mathbb{F}_q^k \setminus \{\boldsymbol{0}\}$. □

Now we introduce Lemma 6.5 which is the binary version of Lemma 6.3 but with an extra restriction on the weight of the words.

Lemma 6.5. *Let k, n, d and $d^{\perp}$ be integers such that*

$$\sum_{i=0}^{d-1} \binom{n}{i} < \frac{1}{4} \cdot 2^{n-k} \text{ and } \sum_{i=0}^{d^{\perp}-1} \binom{n}{i} < \frac{1}{2} \cdot 2^{k}.$$

There exists a binary $[n, k]$ code $\mathcal{C}$ with minimum distance at least d without a word of weight more than $n - d + 1$ such that $\mathcal{C}^{\perp}$ has minimum distance at least $d^{\perp}$.

Proof. Similar to the proof of Lemma 6.3. Let $\mathcal{C}$ denote the set of all binary $[n, k]$ codes. By applying Lemma 6.2 with $E = \{\boldsymbol{e} \in \mathbb{F}_2^n \mid 1 \leq \mathrm{wt}(\boldsymbol{e}) \leq d-1 \text{ or } \mathrm{wt}(\boldsymbol{e}) \geq n-d+1\}$, we infer that the first inequality implies that more than half of the codes in $\mathcal{C}$ contain no element from E. Similarly, the second inequality implies that more than half of the binary $[n, n-k]$ codes have minimum weight at least $d^{\perp}$, and so more than half of the codes in $\mathcal{C}$ have a dual with minimum distance at least $d^{\perp}$. We conclude that there is a code in $\mathcal{C}$ having both desired properties. □

Theorem 6.3 (Gilbert-Varshamov-like bound by Construction 4.5)**.** *Let n, k, l, u, t, μ be positive integers with $u \leq n, 2t < n$ and $l < k$ be such that*

$$\sum_{i=0}^{2t} \binom{n}{i} < \frac{1}{4} \cdot 2^{n-l}, \tag{6.5}$$

$$\sum_{i=0}^{\lfloor \frac{u}{2^{\mu}-1} \rfloor} \binom{n}{i} < \frac{1}{2} \cdot 2^{l}, \tag{6.6}$$

$$\sum_{i=0}^{2t} \binom{n}{i} (2^{\mu}-1)^{i} < 2^{\mu(n-k+1)}, \tag{6.7}$$

Then there exists a $(u, 1, t)$ PSMC of length n over $\mathbb{F}_{2^{\mu}}$ with cardinality $2 \cdot 2^{\mu(k-l-1)} 2^{l(\mu-1)}$.

Proof. By Lemma 6.5, there exists a binary $[n, l]$ code $\mathcal{C}_0$ with minimum distance at least $2t + 1$ for which $\mathcal{C}_0^{\perp}$ has minimum distance at least $\lfloor \frac{u}{2^{\mu}-1} \rfloor + 1$ with the following additional property: if $\boldsymbol{H}_0 \in \mathbb{F}_2^{l \times n}$ is a generator matrix for $\mathcal{C}_0$, then the binary code $\mathcal{C}_{\mu}$ with generator matrix $\begin{bmatrix} \boldsymbol{H}_0 \\ \boldsymbol{1} \end{bmatrix}$ has minimum distance at least $2t + 1$. According to Lemma 6.4, the code $\mathcal{C}_{\mu}$ over $\mathbb{F}_{2^{\mu}}$ with this generator matrix has minimum distance at least $2t + 1$ as well. Lemma 6.1 implies that $\mathcal{C}_{\mu}$ can be extended to an $[n, k]_{2^{\mu}}$ code with minimum distance at least $2t + 1$. The $[n, k]$ code has a generator matrix of the form $\boldsymbol{G} = \begin{bmatrix} \boldsymbol{H}_0 \\ \boldsymbol{G}_1 \\ \boldsymbol{1} \end{bmatrix}$. Application of Proposition 4.1 yields the claim. □

Finite GV bound from trivial construction

Undoubtedly, a $(q-1)$-ary code of length n with minimum distance at least $2t+1$ is a q-ary $(u,1,t)$ PSMC of length n with $u=n$. Combining this observation with the Gilbert bound for a $(q-1)$-ary alphabet, we obtain the following corollary.

Corollary 6.3. *Let $q \geq 3$, and let*

$$M = \left\lceil \frac{(q-1)^n}{\sum_{i=0}^{2t}\binom{n}{i}(q-2)^i} \right\rceil .$$

There is a q-ary $(n,1,t)$ PSMC of length n and cardinality M.

So far we have covered the GV-like bounds for our code constructions for finite length n.

6.3.2 Discussion and Numerical Results

We provide different comparisons between our code constructions and the existence of the code based on Theorem 6.1, Theorem 6.2 and Theorem 6.3. Next, we also compare to the known limits and investigate the trade-off between masking and error-correction as described in Section 5.1.2. Note that we select $q=2^2$ in Figure 6.4 and $q=2^3$ in the rest of the figures (except Figure 6.2 where $q=7$) since we know from Section 3.2.1 that the multi-level cell (MLC) has four levels and the triple-level cell (TLC) has eight levels. Hence, $q=4$ and $q=8$ are practical choices to match these memories from one side and consider Construction 4.5 on the other.

Comparison of Theorem 6.1 for $u \leq q-1$ to other Bounds

Figure 6.2 illustrates the rates of a $(q-1,1,t)$ PSMC obtained from Theorem 4.2 stated in Section 4.5.1 (applying Theorem 6.1 for the special case $u \leq q-1$) for $n=114, q=7$ and $0 \leq t \leq 56$. We show how close explicit BCH codes that contain the all-one word of certain rates R and that can correct designed distances $d \geq 2t+1$ to the achieved rates from Theorem 4.2. We note that the solid red graph matches the dashed-dotted green plot for a few code parameters and overpasses it for $t=39$. We also compare to the classical q-ary GV bound (in dashed black) as well as to reduced alphabet $(q-1)$-ary GV bound (in dashed-dotted blue).

Construction 4.2 in Chapter 4 imposes a generator matrix of an error correction code that discriminates one dimension which is the all-one vector to satisfy the partially defective constraints (defined in Section 3.4.3 and Definition 4.2). Thus, we show upper bounds on the rates that can be obtained from Theorem 4.2 using the Griesmer bound [Gri60] (cf. Section 2.4.2), and the Ball–Blokhuis bound [BB13] (cf. Section 2.4.3) on the size of codes containing the all-one word, i.e., a weight n codeword of dimension $k>1$.

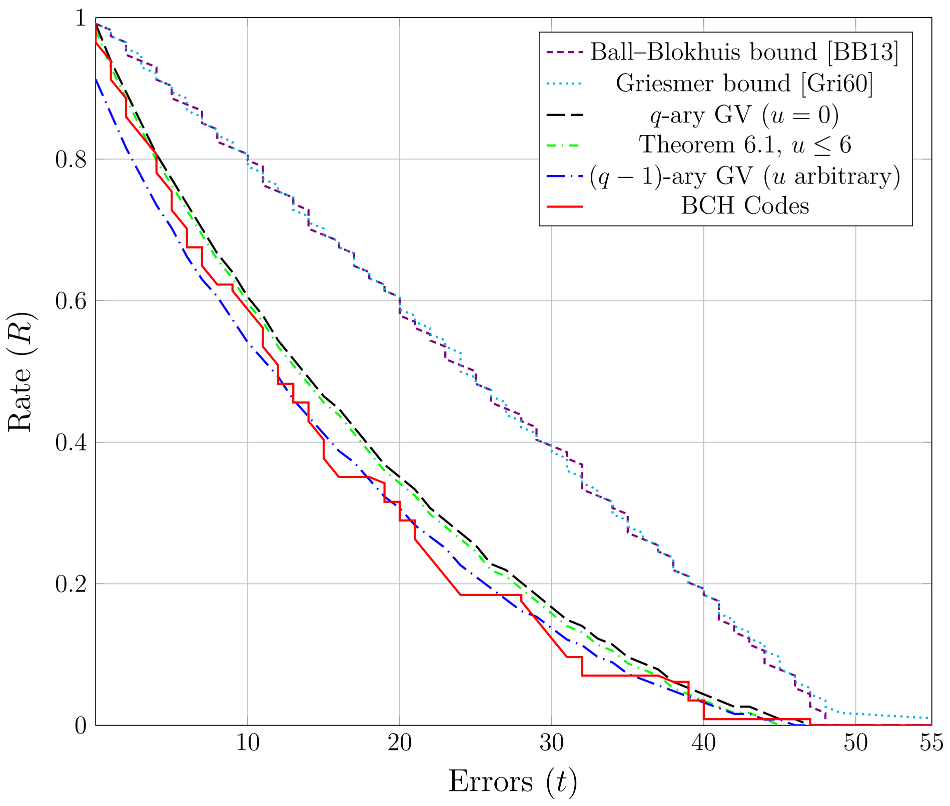


Figure 6.2: Comparison of other upper and lower limits to our derived GV-like bound in Theorem 6.1 taking $n = 114$, $q = 7$, $0 \leq t \leq 56$ and $u \leq q - 1$. The dashed-dotted green curve shows the rates for Theorem 4.2 by Theorem 6.1 for $u \leq q - 1$ in which codes that have the all-one words are considered. This curve for several code parameters matches the red line that shows the rates of BCH codes that contain the all-one word with regard to the designed distances $d \geq 2t + 1$.

Comparison among Theorem 6.2, Theorem 6.3 and $(q-1)$-ary Gilbert-Varshamov bound

We plot the achievable rates ($R = log_q M/n$) as a function of t for different fixed values of u. Figure 6.3 is the resulting plot for $n = 200$, $\mu = 3$ and $q = 2^{\mu}$. It can be seen that the GV-like bound in different ranges of u and t based on Construction 4.4 (cf. Section 4.5.2) improves upon the $(q-1)$-ary GV bound for $u \leq 5$ as depicted in the solid red curve, and improves further (up to $u \leq 20$) based on Construction 4.5 (cf. Section 4.5.2) as shown in the dashed gray line (3rd one from above). The dashed dotted blue curve is used to see what if we map our 2^3 levels such that we avoid the subscript 0 to compare with 7 levels. It is obvious that for $\mu = 3$, the rate loss[1] resulting from using $q-1$ instead of q symbols is already quite small. Note that for $u = 0$ the solid red curve mostly achieves the exact rates obtained from the standard 2^3-ary GV bound for $0 \leq t \leq 80$, and so as for the dashed red curve but for $0 \leq t \leq 47$. For $\mu = 2$, the improvements from Construction 4.4 ($u \leq 10$) and Construction 4.5 ($u \leq 30$) upon a usual $(q-1)$-ary GV bound are more significant as shown in Figure 6.4.

Comparisons between Theorem 6.1 and $(q-1)$-ary Gilbert-Varshamov bound

In Figure 6.5, we compare the GV like bound from Theorem 6.1 for $q = 2^3$ with the conventional GV bound for $q-1 = 7$ shown in dashed blue curve. For $q = 8$, we observe that the conventional $q-1$-ary GV bound is superior to the derived GV-like bound from Theorem 6.1 for $u \geq 40$. However, applying Theorem 6.1 where $u \leq 20$, the traditional $q-1$-ary GV bound is a bad choice. We observe that the dashed-dotted green curve by Theorem 6.1 for ($u \leq 7$ as stated in Remark 6.2) shows the highest rates.

Comparisons between Theorem 6.3 and Theorem 6.1

In Figure 6.6, we compare Theorem 6.3 and Theorem 6.1. Theorem 6.3 is showing higher rates for larger u values, for example taking $u = 40$ and $t = 1$, the rate is $R = 0.87$ from Theorem 6.3 while $R = 0.83$ from Theorem 6.1. It is interesting to note that for $u = 30$ and $t > 10$ Theorem 6.1 is better, and for $u = 10$ and $t > 18$ Theorem 6.1 is as good as Theorem 6.3.

Comparisons of application of Theorem 5.1 vs direct application of Theorem 6.2

For given (u, t), we illustrate the trading $(u+1, t-1)$ in Figure 6.7. For some of t and a few of u values, it is advantageous if the encoder introduces an error in a partially-stuck-at position in order to mask this position. The orange solid curve, for instance, represents the rates that have been determined by Theorem 6.2 for $u = 17$

[1]For $t = 0$, the loss is $1 - \log_8(7) = 0.0642$.

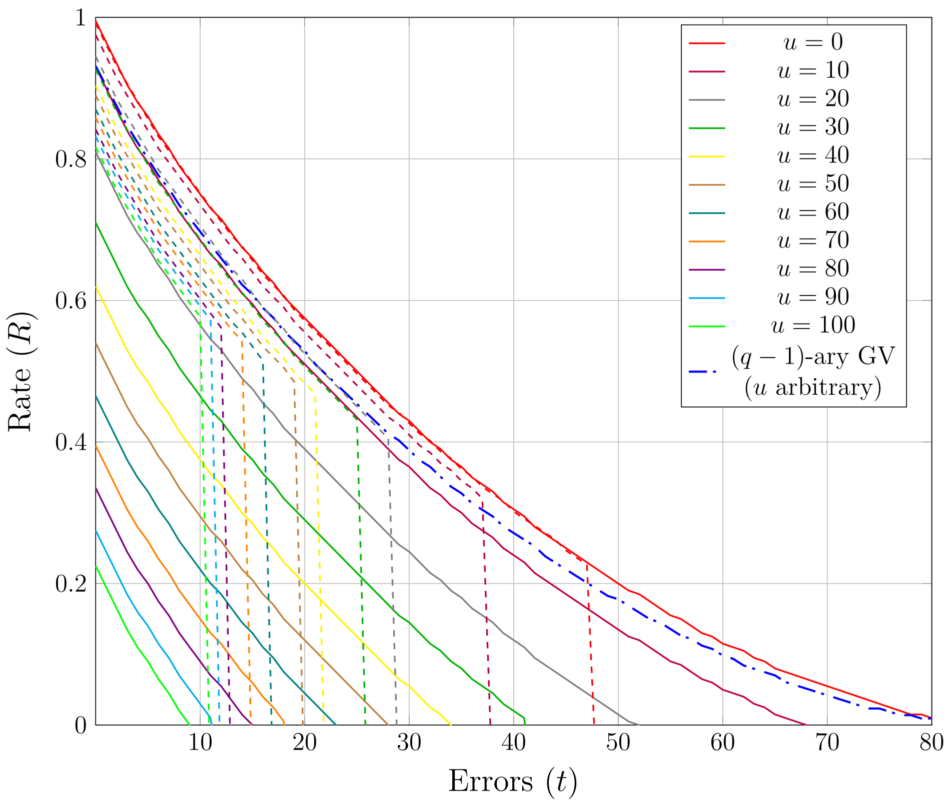


Figure 6.3: The achievable rates $R = \frac{1}{n}\log_{2^3} M$ of GV bounds for different u, t for $n = 200$ and $q = 2^3$ in Theorem 6.2 and Theorem 6.3, where M is the code cardinality. They are also compared to the rates from an ordinary 7-ary GV bound for different t as illustrated in the dashed-dotted blue plot. The solid and the dashed lines represent the derived GV like bounds from Theorem 6.2 and Theorem 6.3, respectively.

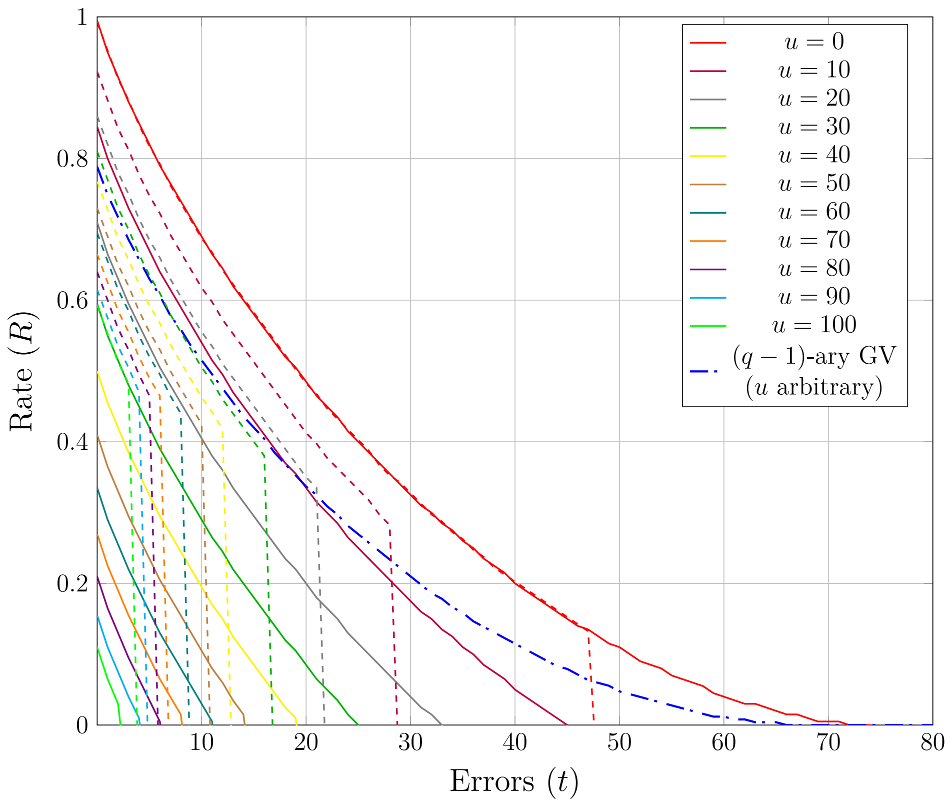


Figure 6.4: The achievable rates $R = \frac{1}{n}\log_{2^2} M$ of GV bounds for different u, t for $n = 200$ and $q = 2^2$ in Theorem 6.2 and Theorem 6.3. They are also compared to the rates from an ordinary 3-ary GV bound for different t as illustrated in the dashed-dotted blue plot. The solid and the dashed lines correspond to the derived GV like bounds by Theorem 6.2 and by Theorem 6.3, respectively.

.

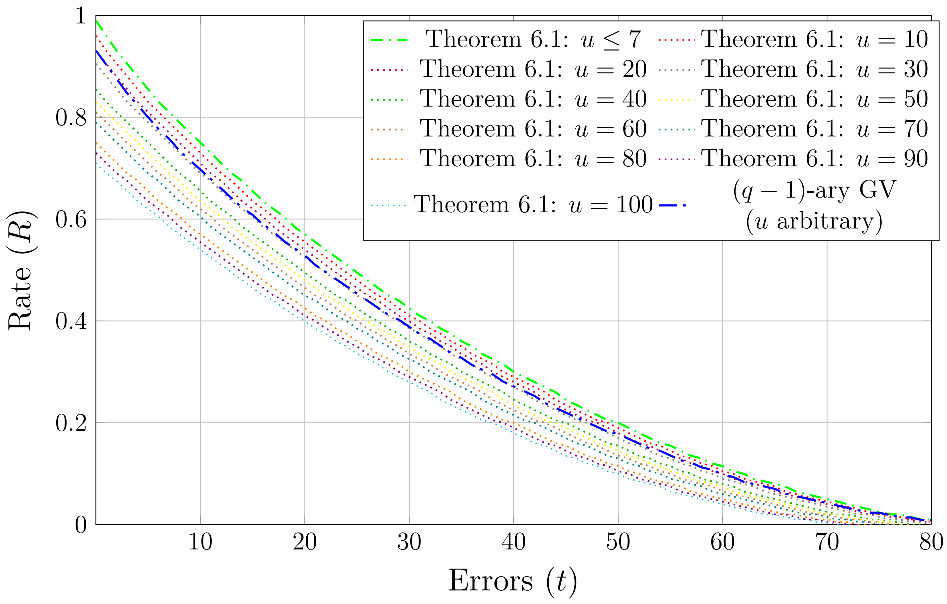


Figure 6.5: The achievable rates $R = \frac{1}{n} \log_{2^3} M$ of GV bounds for different u, t for $n = 200$ and $q = 2^3$ in Theorem 6.1 that are compared to the reduced alphabet conventional $(q-1)$-ary GV bound for different t. The dashed-dotted green curve represents the rates from Theorem 6.1 when $u \leq q-1$.

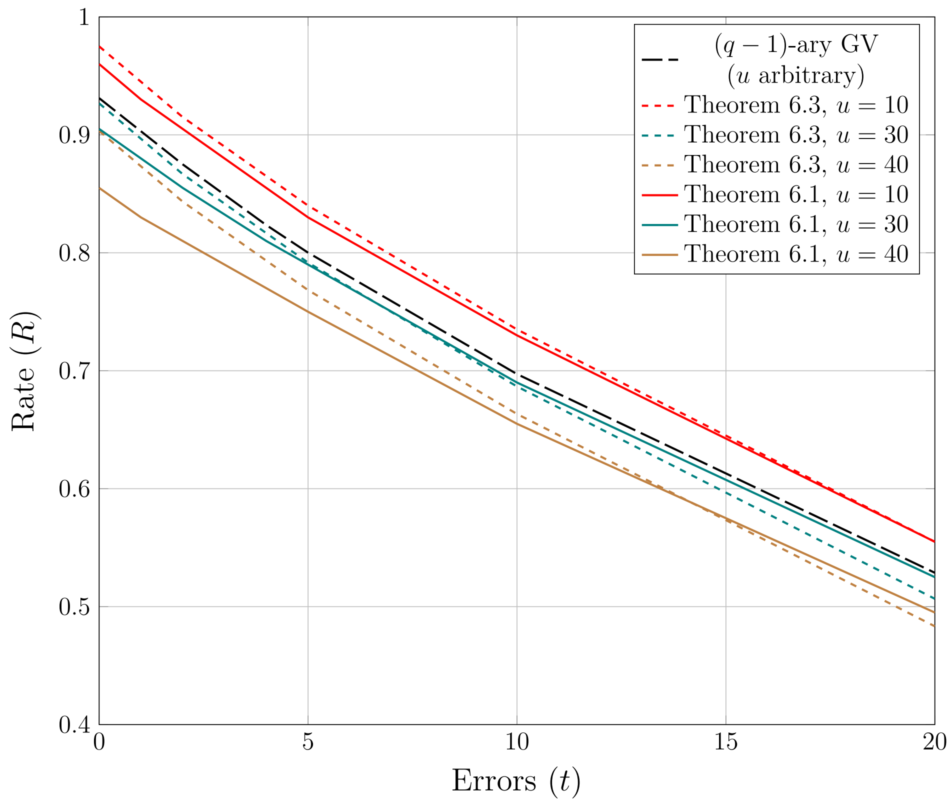


Figure 6.6: The achievable rates $R = \frac{1}{n} \log_q M$ of GV bounds for different $u = \{10, 30, 40\}$, and $t = \{0, 1, 2, 4, 5, 10, 20\}$ for $n = 200$ and $q = 2^3$ in Theorem 6.3 and Theorem 6.1. They are also compared to the rates for $(q-1)$-ary GV bound as shown in dashed black graph.

and $0 \leq t \leq 50$, while the orange dotted sketch highlights the rates for $u = 16$ while $1 \leq t \leq 51$. Due to the exchange such that $u + 1 = 17$ and $0 \leq t - 1 \leq 50$, the orange dotted line slightly fluctuates up and down the rates shown in the orange solid curve for most t values.

Let us describe some points of Figure 6.7 in Table 6.1. Let $\mathcal{C}_{u,t}$ be a code by Theorem 6.2 whose rate is R given in Table 6.1 at u row and t column. Take $\mathcal{C}_{21,15}$ so that its rate $R =$ **0.470**. By applying Theorem 5.1 (see Section 5.1.2) on $\mathcal{C}_{21,15}$, we obtain a code $\mathcal{C}_{22,14}$ of $R = 0.470$. Direct application of Theorem 6.2 yields a $\mathcal{C}_{22,14}$ of rate $R =$ **0.475** as highlighted in Table 6.1. We conclude that in this case, the trade by Theorem 5.1 gives lower rates than taking the same code directly by Theorem 6.2 for given $(u = 22, t = 14)$.

On contrary, for larger t values, Table 6.1 shows that the exchange is beneficial giving higher rates. For example, we start with $\mathcal{C}_{21,41}$ whose $R =$ **0.105**, then applying Theorem 5.1 gives $\mathcal{C}_{22,40}$ of $R = 0.105$ which is greater than $R =$ **0.100** that has been obtained directly by Theorem 6.2 as stated in Table 6.1.

Table 6.1: Table of selected points from Figure 6.7 with slightly lower and higher rates due to trading. All points are from Theorem 6.2.

u \ t	13	14	15	...	40	41	42
16	0.560	0.545	**0.525**	...	0.170	**0.160**	0.150
17	0.545	**0.530**	0.510	...	**0.155**	0.145	0.135
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
21	0.505	0.490	**0.470**	...	0.115	**0.105**	0.095
22	0.490	**0.475**	0.455	...	**0.100**	0.090	0.080
23	0.480	0.465	0.445	...	0.090	0.080	0.070

Comparisons of applications of Theorem 5.1, Lemma 5.1, Lemma 5.2 vs direct application of Theorem 6.3

For the derived GV bound based on Construction 4.5 from Section 4.5.2 obtained by Theorem 6.3, we demonstrate the exchange of a one error correction ability with a single masking capability of a partially stuck cell following Theorem 5.1 (see Section 5.1.2) in Figure 6.8. The solid and dotted lines represent the rates before and after trading, respectively. We also show the exchange by Lemma 5.1 and Lemma 5.2 in which the reduction of the correctable errors by one increases u by $2^{\mu-1}$ and 2^{μ}, respectively. As it is seen in Figure 6.8, every single solid curve by Theorem 6.3 corresponds to multiple values of u due to the floor operation where $\mathcal{C}_0^{\perp}$ has a minimum

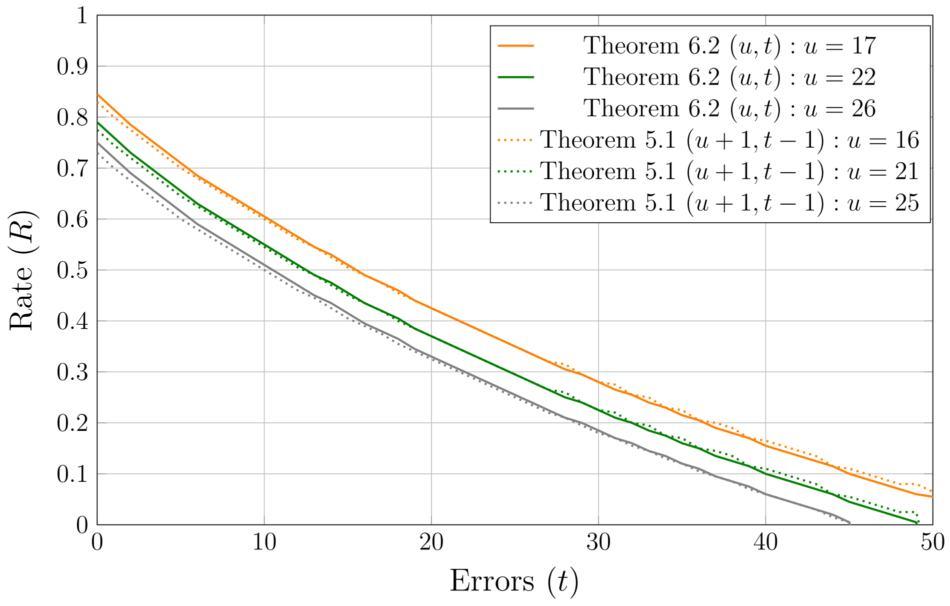


Figure 6.7: The achievable rates $R = \frac{1}{n} \log_q M$ of GV bounds for different $u,$ and t for $n = 200$ and $q = 2^3$ in Theorem 6.2. The solid plots are the rates from the derived GV-like bound and the dotted graphs are the rates after trading $u + 1, t - 1$ by Theorem 5.1.

distance at least $\lfloor \frac{u}{2^{\mu-1}} \rfloor + 1$. For instance, for all $u = 8, 9, 10, 11$ and $\mu = 3$, we obtain: $\lfloor \frac{u}{2^{3-1}} \rfloor + 1 = 3$, and the transition starts for $u = 12$ as $\lfloor \frac{u}{2^{3-1}} \rfloor + 1 = 4$. Similarly by the floor operation there is no change for the larger values of $u = 13, 14, 15$, and so on. Let us discuss the following curves. For $u = 19$, the orange solid curve shows the rates by Theorem 6.3. Exchanging $u + 1$ and $t - 1$ throughout Theorem 5.1 obtains the orange dotted line for $u + 1 = 20$ which lies slightly below the orange solid plot. Hence, the exchange gives lower rates. However, it provides rate $R = 0.380$ for $t = 30$ while direct application of Theorem 6.3 (compared to its corresponding graph which is the solid green curve at $u = 20, 21, 22, 23$) does not.

Now, we apply Lemma 5.1 rather than Theorem 5.1. For the same achieved rates, we observe that the dashed red graph for $u + 2^{3-1} = 23$ shows the exact rates from the orange dotted curve for $u + 1 = 20$. Therefore, it is clear that Lemma 5.1 provides a gain of masking exactly 3 more cells with regard to Theorem 5.1. Further, Lemma 5.1 provides rate at $t = 30$ while Theorem 5.1 stops giving rates at $t \geq 29$. For this graph, we conclude that Lemma 5.1 surpasses Theorem 5.1.

However, if we take $u = 23$ directly by Theorem 6.3, we achieve slightly higher rates. We conclude that Theorem 6.3 can directly estimate the maximum possible masked u cells that can also be achieved applying Lemma 5.1, and can achieve slightly higher rates. On contrary, Theorem 6.3 does not give rates for larger t values while Lemma 5.1 and Theorem 5.1 do that.

On the other hand, as Theorem 6.3 is based on Construction 4.5 that contains a word of weight n, Lemma 5.2 is applicable under the condition that $2(t + \lfloor \frac{u}{q} \rfloor) < d$ (cf. Remark 5.1). Hence, we can achieve higher rates as shown in the dashed-dotted curve while masking up to the same number of u cells, rather employing Lemma 5.1 or Theorem 5.1.

For that we describe some points of Figure 6.8 by Table 6.2. Let $\mathcal{C}_{u,t}$ be a code by Theorem 6.3 whose rate is R given in Table 6.2 at u row and t column. Taking $\mathcal{C}_{19,27}$ gives $\mathcal{C}_{20,26}$ and $\mathcal{C}_{23,26}$ with $R = 0.435$ applying Theorem 5.1 and Lemma 5.1, respectively. In contrary, taking $\mathcal{C}_{19,31}$ is advantageous as there are codes ($\mathcal{C}_{20,30}$ by Theorem 5.1 and $\mathcal{C}_{23,30}$ by Lemma 5.1) with $R = 0.380$ while direct application of Theorem 6.3 cannot provide these codes as highlighted in green with "**None**". Now, we apply Lemma 5.2 on a code obtained by Theorem 6.1 for ($u = 7, t = 27$) to obtain the code $\mathcal{C}_{15,26}$ of rate $R = 0.465$ that satisfies $2(26 + \lfloor \frac{15}{8} \rfloor) < 55$. The achieved rate is higher compared to $\mathcal{C}_{15,26}$ of $R = 0.460$ that is directly obtained by Theorem 6.3, or applying Theorem 5.1 on $\mathcal{C}_{14,27}$ to obtain $\mathcal{C}_{15,26}$ of $R = 0.445$, or using Lemma 5.1 on $\mathcal{C}_{11,27}$ to obtain $\mathcal{C}_{15,26}$ of $R = 0.456$. This result does not mean that application Lemma 5.2 on a code obtained by Theorem 6.1 always provides higher code rates for the same parameters u, t (see Figure 6.6).

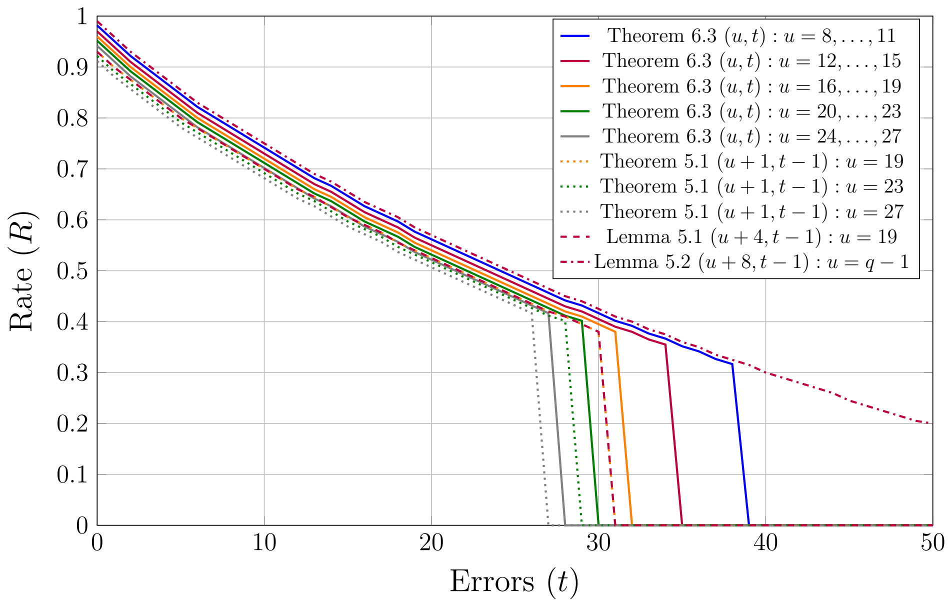


Figure 6.8: The achievable rates $R = \frac{1}{n}\log_q M$ of GV bounds for different u, and t for $n = 200$ and $q = 2^3$ in Theorem 6.3. The solid plots are the rates from the derived GV like bound and the dotted graphs are the rates after trading $u+1, t-1$ by Theorem 5.1. We also show the exchange by Lemma 5.1 and Lemma 5.2 in which the reduction of the correctable errors by one increases u by $2^{\mu-1}$ and 2^{μ}, respectively.

Table 6.2: Table of selected points from Figure 6.8. All points are from Theorem 6.3.

u \ t	26	27	28	29	30	31	32
11	0.471	0.456	0.441	0.431	0.416	0.401	0.391
12	0.460	0.445	0.430	0.420	0.405	0.390	0.380
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
14	0.460	0.445	0.430	0.420	0.405	0.390	0.380
15	0.460	0.445	0.430	0.420	0.405	0.390	0.380
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
19	0.450	**0.435**	0.420	0.410	0.395	**0.380**	None
20	**0.441**	0.426	0.411	0.401	**None**	None	None
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
23	**0.441**	0.426	0.411	0.401	**None**	None	None

6.3.3 Asymptotic Gilbert–Varshamov-type Bound on PSMCs

In this section, we present the asymptotic version of the GV bounds from the previous section. That is, we provide lower bounds on the achievable rates of a q-ary $(u, 1, t)$ PSMCs in the regime that the code length n tends to infinity, and the number u of partially-stuck-at cells and the number t of random errors both grow linearly in n.

In the subsequent sections, we use Lemma 2.2 (cf. Section 2.4.6) that estimates the volume of a Hamming ball using the q-ary entropy function.

Asymptotic Gilbert-Varshamov-like bound by Construction 4.2

We observed in Section 6.3.1 that Theorem 6.1 provides (note Remark 6.2) the existence of PSMCs from Construction 4.2 (cf. Section 4.5.1). Then, the asymptotic version of Theorem 6.1 is expressed as follows.

Theorem 6.4. *Let q be a prime power. Let $0 \leq \tau, \upsilon < 1$ be such that*

$$2(\tau + \frac{\upsilon}{q}) < 1 - \frac{1}{q}.$$

For sufficiently large n, there exists an $(\lfloor \upsilon n \rfloor, 1, \lfloor \tau n \rfloor)$ PSMC of length n and rate at least

$$1 - h_q(2(\tau + \frac{\upsilon}{q})) - \frac{2}{n}.$$

Proof. Let n be a positive integer such that $\lceil nh_q(2(\tau + \frac{\upsilon}{q})\rceil < n$. Let $t = \lfloor \tau n \rfloor$ and $u = \lfloor \upsilon n \rfloor$. Take $k = n - \lceil nh_q(2\tau + 2\frac{\upsilon}{q})\rceil$. Lemma 2.2 (see Section 2.4.6) implies that $\mathrm{Vol}_q(n, 2t + 2\lfloor \frac{u}{q} \rfloor) \leq q^{n-k}$, and so, according to Theorem 6.1, there is a q-ary $(u, 1, t)$ PSMC of length n with rate $\frac{k-1}{n} \geq 1 - h_q(2(\tau + \frac{\upsilon}{q})) - \frac{2}{n}$. □

Asymptotic Gilbert-Varshamov-like bound by Construction 4.4

Theorem 6.2 (cf. Section 6.3.1) was found as GV-like bound from Construction 4.4 (stated in Section 4.5.2). We state its asymptotic version in the following theorem.

Theorem 6.5 (Asymptotic Gilbert-Varshamov-like bound from Theorem 6.2)**.** *Let q be a prime power. Let υ, τ be such that*

$$0 < \upsilon, 2\tau < 1 - \frac{1}{q} \text{ and } h_q(\upsilon) + h_q(2\tau) < 1.$$

For sufficiently large n, there exists a q-ary $(\lfloor \upsilon n \rfloor, 1, \lfloor \tau n \rfloor)$ PSMC of length n and rate at least

$$1 - h_q(2\tau) - h_q(\upsilon) - \frac{4\log_q(2) + 2}{n}.$$

Proof. Let n be a positive integer. Write $u = \lfloor \upsilon n \rfloor$ and $t = \lfloor \tau n \rfloor$. Then $\mathrm{Vol}_q(n, u - q + 2) \leq \mathrm{Vol}_q(n, u)$. Hence, by setting

$$l = \lceil nh_q(\upsilon) + 2\log_q(2)\rceil,$$

Lemma 2.2 implies that (6.2) is satisfied.

Similarly, by setting

$$k = n - \lceil nh_q(2\tau) + 2\log_q(2)\rceil,$$

it is ensured that (6.3) is satisfied.
According to Theorem 6.2, there is a q-ary $(u, 1, t)$ PSMC of length n and size q^{k-l}, so with rate $\frac{k-l}{n}$. The choices for k and l show that the theorem is true. □

Remark 6.4. *Theorem 6.5 in fact holds for classical stuck-at cells instead of stuck-at-*1 *errors, as follows from considering the generalization of Theorem 6.2 in Proposition 4.2, i.e., Heegard's construction [Hee83].*

Asymptotic Gilbert-Varshamov-like bound by Construction 4.5

Theorem 6.3, expressed in Section 6.3.1, is the GV-type bound regarding Construction 4.5, presented in Section 4.5.2. The following sequel is its GV-like boundary in the asymptotic model.

Theorem 6.6 (Asymptotic Gilbert-Varshamov-like bound from Theorem 6.3)**.** *Let μ be a positive integer, and let υ and τ be such that*

$$0 \leq \frac{\upsilon}{2^{\mu-1}} < \frac{1}{2}, 0 < 2\tau < \frac{1}{2}, \text{ and } h_2(\frac{\upsilon}{2^{\mu-1}}) + h_2(2\tau) < 1.$$

For sufficiently large n there is a 2^μ-ary $(\lfloor \upsilon n \rfloor, 1, \lfloor \tau n \rfloor)$ PSMC of length n and rate at least

$$1 - h_{2^\mu}(2\tau) - \frac{1}{\mu} h_2(\frac{\upsilon}{2^{\mu-1}}) - \frac{2}{n} - \frac{3}{\mu n}.$$

Proof. For notational convenience, we set $\upsilon_0 = \frac{\upsilon}{2^{\mu-1}}$ and $\eta = 1 - h_2(2\tau) - h_2(\upsilon_0)$. Note that $\eta > 0$.
Let n be a positive integer satisfying $n \geq \frac{7}{\eta}$, and let $u = \lfloor \upsilon n \rfloor$, $u_0 = \lfloor \frac{u}{2^{\mu-1}} \rfloor$ and $t = \lfloor \tau n \rfloor$. We set

$$l = \lceil n h_2(\upsilon_0) \rceil + 3.$$

Lemma 2.2 implies that (6.6) is satisfied. Moreover, as

$$n - l - 3 \geq n - n h_2(\upsilon_0) - 7 = n h_2(2\tau) + n\eta - 7 \geq n h_2(2\tau),$$

Lemma 2.2 implies that (6.5) is satisfied.
We set

$$k = n - \lceil n h_{2^\mu}(2\tau) \rceil.$$

Lemma 2.2 implies that (6.7) is satisfied.
According to [GRS19, Corollary 3.3.4], we have that $h_{2^\mu}(2\tau) \leq h_2(2\tau)$, and so

$$k - l \geq n - n h_2(2\tau) - 1 - n h_2(\upsilon_0) - 4 = n\eta - 5 \geq 2.$$

Theorem 6.3 implies the existence of a 2^μ-ary $(u, 1, t)$ PSMC of length n with size $2 \cdot 2^{\mu(k-1)} 2^{-l}$, i.e., its rate is

$$\frac{k-1}{n} - \frac{l-1}{\mu n} \geq 1 - h_{2^\mu}(2\tau) - \frac{1}{\mu} h_2(\upsilon_0) - \frac{2}{n} - \frac{3}{\mu n}. \qquad \square$$

Section 6.3.1 showed Corollary 6.3 as the GV-type bound corresponding to the $(q-1)$-ary codes being used as PSMCs. Below, we state the asymptotic type of Corollary 6.3.

Theorem 6.7 (Asymptotic Gilbert-Varshamov bound from Corollary 6.3)**.** *Let $q \geq 3$. For each positive integer n and each τ with $0 \leq 2\tau < 1 - \frac{1}{q-1}$, there exists a q-ary $(n, 1, \lfloor \tau n \rfloor)$ PSMC of length n and rate at least*

$$(1 - h_{q-1}(2\tau)) \cdot \log_q(q-1).$$

Proof. Let $t = \lfloor \tau n \rfloor$. Corollary 6.3 implies the existence of a q-ary $(n, 1, t)$ PSMC of length n and cardinality M satisfying

$$M \geq \frac{(q-1)^n}{V_{q-1}(n, 2t)} \geq (q-1)^{n(1-h_{q-1}(2\tau))},$$

where the last inequality holds by Lemma 2.2. □

6.3.4 Discussion and Analytical Results

In the remainder of this chapter, we state the results of the analytical comparisons of the asymptotic GV bounds from Theorems 6.4, 6.5 and 6.6, ignoring the terms that tend to zero for increasing n. We will use the following lemma.

Lemma 6.6. *Let $q \geq 2$ be an integer. If $0 \leq x, y$ are such that $x + y \leq 1$, then*

$$h_q(x+y) \leq h_q(x) + h_q(y).$$

Proof. Let $0 \leq y < 1$, and consider the function $f_y(x) = h_q(x+y) - h_q(x) - h_q(y)$ on the interval $[0, 1-y]$. Clearly, $f'_y(x) = h'_q(x+y) - h'_q(x) \leq 0$, where the inequality follows from the fact that the second derivative of h_q is non-negative. Hence, $f_y(x) \leq f_y(0) = 0$ for each $x \in [0, 1-y]$. □

Comparisons between Theorem 6.4 and Theorem 6.5

Proposition 6.2. *If υ, τ and q are such that the conditions of Theorem 6.4 and Theorem 6.5 are met, then the rate guaranteed by Theorem 6.4 is at least equal to the code rate guaranteed by Theorem 6.5.*

Proof. Assume τ and υ are such that the conditions of Theorem 6.4 and Theorem 6.5 are satisfied, that is, such that $2\tau + 2\frac{\upsilon}{q} < 1 - \frac{1}{q}$ and $h_q(\upsilon) + h_q(2\tau) < 1$. By invoking Lemma 6.6, we see that

$$h_q(2\tau + 2\frac{\upsilon}{q}) \leq h_q(2\tau) + h_q(2 \cdot \frac{\upsilon}{q}) \leq h_q(2\tau) + h_q(\upsilon),$$

where the final inequality holds as $q \geq 2$ and h_q is monotonically increasing on $[0, 1-\frac{1}{q}]$. As a consequence, the code rate guaranteed by Theorem 6.4 is at least equal to the code rate guaranteed by Theorem 6.5. □

Comparisons between Theorem 6.5 and Theorem 6.6

Proposition 6.3. *If υ, τ and $q = 2^\mu$ are such that the conditions of Theorem 6.5 and Theorem 6.6 are met, then the code rate guaranteed by Theorem 6.6 is at least equal to the code rate guaranteed by Theorem 6.5.*

Proof. Assume that the conditions of Theorem 6.5 and Theorem 6.6 are satisfied. The difference between the rate of Theorem 6.6 and of Theorem 6.5 equals

$$h_{2^\mu}(\upsilon) - \frac{1}{\mu} h_2(\frac{\upsilon}{2^{\mu-1}}). \tag{6.8}$$

According to the conditions of Theorem 6.5, $\upsilon \le 1 - \frac{1}{2^\mu}$, and so $h_{2^\mu}(\upsilon) \ge h_2(\frac{\upsilon}{2^{\mu-1}})$. As $h_{2^\mu}(x) = \frac{1}{\mu} h_2(x) + x \log_{2^\mu}(2^\mu - 1)$, the difference in (6.8) is non-negative. That is, Theorem 6.6 is better than Theorem 6.5. □

We note that the requirement $2\tau < \frac{1}{2}$ from Theorem 6.6 is stricter than the requirement $2\tau < 1 - \frac{1}{2^\mu}$ from Theorem 6.5. That is, there are pairs (τ, υ) for which Theorem 6.5 is applicable, but Theorem 6.6 is not.

Comparison of Theorem 6.4 and Theorem 6.6 is more complicated. We have the following partial result.

Comparisons between Theorem 6.4 and Theorem 6.6

Proposition 6.4. *Let $\upsilon, \tau > 0$ and $q = 2^\mu$ be such that the conditions of Theorem 6.4 and Theorem 6.6 are met. If υ is sufficiently small, then the rate guaranteed by Theorem 6.4 is larger than the rate guaranteed by Theorem 6.6.*

Proof. Assume that τ and υ are such that the conditions of Theorem 6.4 and of Theorem 6.6 are satisfied, that is, $2\tau + 2\frac{\upsilon}{2^\mu} < 1 - \frac{1}{2^\mu}$,

$$0 \le \upsilon \le 2^{\mu-2},\ 0 \le 2\tau \le \frac{1}{2} \text{ and } h_2(\frac{\upsilon}{2^{\mu-1}}) + h_2(2\tau) < 1.$$

Let $f_\mu(\tau, \upsilon_0)$, where $\upsilon_0 = \frac{\upsilon}{2^{\mu-1}}$, be the bound from Theorem 6.4 minus the bound from Theorem 6.6, that is

$$f_\mu(\tau, \upsilon_0) = h_{2^\mu}(2\tau) + \frac{1}{\mu} h_2(\upsilon_0) - h_{2^\mu}(2\tau + \upsilon_0).$$

The definition of the entropy function implies that for any $x \in [0,1]$

$$h_{2^\mu}(x) = \frac{1}{\mu}\left(h_2(x) + x \log_2(2^\mu - 1)\right). \tag{6.9}$$

Applying (6.9) , we infer that

$$\mu f_\mu(\tau, \upsilon_0) = h_2(2\tau) + h_2(\upsilon_0) - h_2(2\tau + \upsilon_0) - \upsilon_0 \log_2(2^\mu - 1). \tag{6.10}$$

In particular, $\mu f_\mu(0, \upsilon_0) = -\upsilon_0 \log_2(2^\mu - 1) \le 0$.

So for $\tau = 0$, Theorem 6.6 is better than Theorem 6.4. It follows from Lemma 6.6 that the three leftmost terms in (6.10) form a non-negative number. The subtraction of the fourth term, however, can result in a negative function value, especially for large μ.

Example 6.1 (Numerical example). $\mu f_\mu(0.055, 0.11) = 2 \cdot h_2(0.11) - h_2(0.22) - 0.11 \log_2(2^\mu - 1) \approx 0.23397 - 0.11 \log_2(2^\mu - 1)$ *is positive for* $\mu \leq 2$ *and negative otherwise.* ▶

We now prove Proposition 6.4. That is, we show that for $\tau > 0$ and υ_0 sufficiently small, $f_\mu(\tau, \upsilon_0) > 0$. This follows from the Taylor expansion of $\mu f_\mu(\tau, \upsilon_0)$ around $\upsilon_0 = 0$. Indeed, $f_\mu(\tau, 0) = 0$, and $h_2'(x) \to \infty$ if $x \downarrow 0$. □

6.4 Bounds on Polyalphabetic Codes with Finite Alphabets

Mixed codes, which are error-correcting codes in the Cartesian product of different-sized spaces (cf. Section 2.5), model degrading storage systems well, e.g., unreliability due to partially stuck-at scenario (cf. Section 3.4.3). Furthermore, recall that Section 6.2 exhibits that upper bounds on the size of polyalphabetic codes are also upper bounds on the partially stuck-at codes. Theretofore, we were motivated to further investigate different upper/lower bounds on polyalphabetic codes of *finite alphabets*, i.e., unlike the case of unbounded alphabet sizes in [Sid$^+$05]. Our recent work [YAPW23] focuses on the case of finite alphabets, and generalizes the Gilbert–Varshamov, sphere-packing, Elias-Bassalygo (in [Bas65], and reported in [Joh63]), and first linear-programming [Lev98; MRRW77] bounds to that setting. In the latter case, our proof is also the *first* for the non-symmetric *monoalphabetic* q-ary case using Navon and Samorodnitsky's Fourier-analytic approach [NS09].

To avoid distracting the reader from the main purpose of this dissertation, we omit these results and refer the interested reader to our paper [YAPW23]. Nevertheless, we summarize its main contributions as follows. Unlike [Sid$^+$05] that proposes a straightforward expression for sphere size, containing an exponential number of terms (cf. (2.36) in Section 2.5.1), we provide a recursive formula for the size of spheres that allows efficient computation of exact sizes in any given case, resulting in said bounds on code sizes. We also derive closed-form upper and lower bounds on the size of spheres, yielding *asymptotic* expressions for the size of *balls* which readily give the *asymptotic* Gilbert-Varshamov and sphere-packing bounds. Finally, we develop the equivalence of the Elias-Bassalygo bound and the first linear-programming bound for mixed codes, which are *tighter* for codes with some minimum distances when alphabet sizes are bounded, in contrast to a known bound (restated in Theorem 2.9 and [Etz22, Corollary 2.15], which curiously develop the same bound in this context).

7 Conclusion and Outlook

THIS dissertation studied various characteristics of algebraic coding theory focusing on concepts related to memory with defects.

In Chapter 2, we explored and summarized the channel coding concepts and their regarded principles. We referred to, in Chapter 3, the state-of-the-art work employing error-correcting codes for memories with defects (as they are point-to-point communication schemes with a discreet memoryless channel). We also defined in the same chapter possible sorts of such memories, their potential noise, and error types.

In Chapter 4, code constructions for non-volatile memories with partial defects have been proposed. Our constructions can handle both: partial defects (also called partially stuck cells) and random substitution errors, and require less redundancy symbols for the number of partially stuck cells $u > 1$ and alphabet sizes $q > 2$ than the known constructions for stuck cells. Compared to error-free masking of partially stuck cells, our achieved code sizes coincide with those in [WY16], or are even larger. We summarize our constructions and the previous works on partially/fully stuck cells in Table 4.3.

In Chapter 5, we have shown that it can be advantageous to *intentionally* introduce errors in some partially stuck cells in order to satisfy the stuck-at constraints. For the general case that is applicable for all of our constructions, we have shown how to replace any $0 \leq j \leq t$ errors by j masked partially stuck cells. This result has been improved for our construction based-binary, and further enhanced by another method for introducing errors in the partially stuck locations (cf. Example 5.2). We gain (e.g., for $j = 1$) exactly $2^{\mu-1}$ and 2^{μ} (under the condition that $2(t + \lfloor \frac{u}{2^{\mu}} \rfloor) < d$) additional masked partially stuck cells applying the former improvements, respectively. So far, determining if introducing errors in partially stuck cells is advantageous or not can only be done numerically.

We also derived upper and lower limits on the size of our constructions in Chapter 6. Our sphere-packing-like bound for the size of (Σ, t) PSMCs (partially-stuck-at-masking codes) has been compared to the usual sphere-packing upper bound, and for the case of no errors ($t = 0$) to [WY16, Theorem 2].

We have numerically compared our Gilbert–Varshamov-type bounds, for given (u, t), to each other and to $(q-1)$-ary codes. For $u \leq q-1$, we formulated a theorem to state the existence of $(u, 1, t)$ PSMCs with rates that almost match the ones from the usual q-ary GV bound. Moreover, up to $u = 20$ for $q = 8$, we show that application of this theorem is better than using $(q-1)$-ary code as mentioned in [WY16, Section III]. On the other hand, for $q = 4$ and $u = 10$, our other derived theorems for GV-like bound require less redundancy than $(q-1)$-ary code.

Treating partially stuck cells as errors can reduce the necessary redundancy for some parameters. Our work utilized the formulated GV-type bounds to demonstrate that sacrificing a single error correctability can lead to higher code rates and more masked cells.

In the asymptotic regime of our GV-like bounds, several bounds are strong competitors, but determining which one is the best choice is not a simple task. The analytical comparison between two of the competing bounds, referred to as Theorem 6.4 and Theorem 6.6, is particularly challenging. Numerical results presented earlier in Chapter 6 provide additional confirmation of this challenge. Finally, we shortly summarized our derived upper/lower bounds on polyalphabetic codes considering *limited* allowed alphabets in some coordinates. Our formulated closed-form expressions on the size of spheres readily express the (asymptotic) Gilbert-Varshamov and sphere-packing bounds.

The open problems in coding for partially stuck memory cells include finding more efficient coding schemes using alternative metrics (instead of the Hamming metric) and exploring the potential of combining different coding models. Further research involves using array codes to repair *burst* errors or utilizing polyalphabetic mother codes to propose substitute coding algorithms.

A Appendix

The appendix covers some of our expectations, observations, remarks, alternatives, and earlier findings. Appendix A.1 comments on array codes in storage applications. Then Appendix A.2 provides an observation and a remark on polyalphabetic mother code-based construction. Short comparisons between other coding-built approaches for unreliable memories and our code constructions have also been listed in Appendix A.3. An alternative proof of Proposition 4.2, which is located in Chapter 4, is given in Appendix A.4. Finally, some of our interesting earlier results on the GV-like bound based on our constructions in Section 4.5.2 have been stated in Appendix A.5.

A.1 Observation on Array Codes in Storage Applications

In the literature on coding theory, there are also *array codes*, which are generalization of *scalar* codes (also called *vector* codes[1]) where their elements are matrices, i.e., codewords are represented in columns instead of single coordinates. In these codes, a codeword of a linear code over $\mathbb{F}_p$ (for prime p) represents a row of the *array codeword* over $\mathbb{F}_{p^\mu}$ (for $\mu > 1$), i.e., applying Definition A.1 given below. The remarkable property of array codes is that they are seen as codes over an extension field with the same parameters, fulfilling Remark 2.2.

Array codes are famous for their usability in storage applications. For example, a flash memory unit (see Section 3.2.1) commonly stores any of the 2^μ values as a binary column representation, where $\mu \geq 1$ is referred to as the *subpacketization.* Each bit located on a different page belongs to a certain 2^μ-ary memory cell and holds either 0 or 1. Any vector $\boldsymbol{m} \in \mathbb{F}_{2^\mu}^k$ can be represented as a matrix $\boldsymbol{M} \in \mathbb{F}_2^{\mu \times k}$ applying the following definition.

[1]In Section 2.3, linear codes whose codewords are *vectors* of equal length n over a given finite (extension) field have been introduced (cf. (2.4)).

Definition A.1 (Mapping to Base Field)**.** *Assume a vector $\boldsymbol{a} \in \mathbb{F}_{p^\mu}^n$. Denote a basis of $\mathbb{F}_{p^\mu}$ over $\mathbb{F}_p$ as $\mathcal{B}$ and let an order of this basis $\boldsymbol{\beta} = (\beta_0, \beta_1, \dots, \beta_{\mu-1})$ be fixed. Then the bijective map $\mathcal{Q} : \mathbb{F}_{p^\mu}^n \mapsto \mathbb{F}_p^{\mu \times n}$ that expresses the extension of $\boldsymbol{a}$ over the base field is as follows.*

$$\boldsymbol{a} = (a_0, a_1, \dots, a_{n-1}) \mapsto \boldsymbol{A} = \begin{pmatrix} A_{0,0} & A_{0,1} & \dots & A_{0,n-1} \\ A_{1,0} & A_{1,1} & \dots & A_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m-1,0} & A_{m-1,1} & \dots & A_{m-1,n-1} \end{pmatrix},$$

where $\boldsymbol{A} \in \mathbb{F}_p^{\mu \times n}$ is a matrix that is formed such that

$$a_j = \sum_{i=0}^{\mu-1} A_{i,j} \beta_i, \ \ \forall j \in [n].$$

Application $\mathcal{Q} : \mathbb{F}_{p^\mu}^n \mapsto \mathbb{F}_p^{\mu \times n}$ to a single element $a \in \mathbb{F}_{p^\mu}$ turns it to a vector $\boldsymbol{a} = (a_0, a_1, \dots, a_{\mu-1}) \in \mathbb{F}_p^\mu$, i.e., $a \in \mathbb{F}_{p^\mu}^1 \mapsto \boldsymbol{a} \in \mathbb{F}_p^{\mu \times 1}$. Furthermore, any element $a \in \mathbb{F}_{p^\mu}$ can be uniquely represented as $a = \sum_{i=0}^{\mu-1} a_i \beta_i$ where $a_i \in \mathbb{F}_p$ for all i. The inverse mapping, $\mathcal{Q}^{-1} : \mathbb{F}_p^{\mu \times n} \mapsto \mathbb{F}_{p^\mu}^n$, also holds by Definition A.1.

Definition A.1 is essential to understand how to represent *codewords* (cf. Section 2.3) in either a vector over the extension field (e.g., over $\mathbb{F}_{2^\mu}^n$) or a matrix over the base field (e.g., over $\mathbb{F}_2^{\mu \times n}$).

Homogeneous interleaved codes [MK90; KL97], types of array codes, have another significant feature which is increasing the decoding radius beyond $\lfloor \frac{d-1}{2} \rfloor$. They decode up to $d-2$ errors (see the *list decoder* in Section 2.3.3) with high probability [Hol+21; RPW21]. We do not consider these codes in this dissertation. However, for interested researchers, these codes class could provide an extension for our work that regards only unique decoding radius (cf. Section 2.3.3).

A.2 Observation on Polyalphabetic Codes

Nested BCH codes in Section 2.3.6 build a connection to polyalphabetic codes (cf. Section 2.5), i.e., codes over specific fields can be used to construct polyalphabetic codes over alphabets of sizes at most the sizes of these fields. The following remark gives the foundation for the proposal.

Remark A.1. *Linear codes over $\mathbb{F}_q$ for prime power $q = p^\mu$ are mother codes for polyalphabetic codes (defined in Section 2.5) of equal alphabets in each coordinate i, i.e., $\{q_i = p^h, \forall i\}$ since their subfield subcodes are codes over $\mathbb{F}_{p^h}$ for $1 \leq h \leq \mu$ (e.g., nested BCH codes Section 2.3.6).*

This dissertation did not include coding schemes using polyalphabetic mother codes for partially stuck cells case. It could be considered an open problem for further analysis. Nevertheless, one could proceed to propose a construction as follows.

An encoder for partially stuck cells scenario could use a mother code over p^h, e.g., any subfield subcode over p^h of Reed-Solomon (RS) codes [Rot06, Chapter 5] over p^μ for prime p and $1 \leq h < \mu$. The encoder knows the allowed alphabets at each position, i.e., any reduced alphabets in some positions are likening as partially stuck cells. Thus, following the encoding procedure in [Sid+05, Section IV], the output vector could be produced such that some of its coordinates (i.e., the information portion) are from a reduced *information space* satisfying the partially stuck cells, given a restriction on the partially stuck cell locations. On the receiver side, the decoder uses the mother code over the alphabet p^h and decodes, e.g., a nested BCH code (cf. Section 2.3.6), to reconstruct the encoded message.

A.3 Remarks on Coding-based Methods for Non-Volatile Memories

The following remarks can shortly summarize the relative comparisons between partially stuck memory cells (PSMC) codes (cf. Chapter 4), write once memory (WOM) codes (cf. Section 3.4.1), and rank modulation (cf. Section 3.4.2).

Remark A.2 (PSMC and WOM). *WOM codes (cf. Section 3.4.1) have a limited number of* writes *that can be performed before the need to decrease the cell levels. In contrast, this limitation in the number of writes does not exist in the partially stuck memory construction, i.e., the cell level can be decremented as long as the conditions for partially stuck cells are inviolable. Furthermore, the constructions for WOM codes, e.g., in [SC19b], are suited for a quite short code length and limited alphabet sizes, namely for* $n = 2$ *and* $q = 8$*. Contrary to these WOM codes, PSMC codes in Chapter 4 are applicable for memories with a huge number of cells using long code block lengths, and alphabet sizes satisfying* $q < n$*.*

Remark A.3 (PSMC and Rank Modulation). *Although rank modulation (cf. Section 3.4.2) addresses asymmetric drifts of cell levels, errors can still occur since the cell levels do not always drift at the same rate [JSB08]. Unlike rank modulation, partially stuck memory constructions do not consider reordering or permutation cell levels to store information but rather the relative cell values. They handle all cell-level drifts as long as they cause partially stuck scenarios or random errors. Compared with [JSB08], which proposes single-error correction codes, all constructions in Chapter 4 deal with any number of partially defective cells and errors. Furthermore, the erasing step could be* always *avoided as it is the zero value (i.e., the partially stuck at state).*

A.4 An Alternative Proof for Generalization to Arbitrary Partially Stuck Levels

We start with Lemma A.1, and then we give an alternative proof of Proposition 4.2 from Chapter 4.

Lemma A.1. *Let $\boldsymbol{M} \in \mathbb{F}_q^{m\times n}$ be such that each column of $\boldsymbol{M}$ has at least one non-zero entry. Let $\boldsymbol{s} \in \mathbb{F}_q^n$. For each $\boldsymbol{w} \in \mathbb{F}_q^n$, there is a $\boldsymbol{v} \in \mathbb{F}_q^m$ such that*

$$\left|\left\{i \in [n] \mid w_i + (\boldsymbol{vM})_i \geq s_i\right\}\right| \geq n - \left\lfloor \frac{1}{q}\sum_{i=0}^{n-1} s_i \right\rfloor .$$

Proof. We define the set S as

$$S = \left\{(i, \boldsymbol{v}) \in [n] \times \mathbb{F}_q^m \mid w_i + (\boldsymbol{vM})_i \geq s_i\right\}.$$

Clearly, there is $\boldsymbol{v} \in \mathbb{F}_q^m$ such that

$$\left|\left\{i \in [n] \;\middle|\; w_i + (\boldsymbol{vM})_i \geq s_i\right\}\right| \geq \left\lceil \frac{|S|}{q^m} \right\rceil . \tag{A.1}$$

Let $i \in [n]$. As the i-th column of $\boldsymbol{M}$ has a non-zero entry, for each $y \in \mathbb{F}_q$ there are exactly q^{m-1} vectors $\boldsymbol{x} \in \mathbb{F}_q^m$ such that $(\boldsymbol{xM})_i = y$. As a consequence,

$$\left|\left\{\boldsymbol{v} \in \mathbb{F}_q^m \mid w_i + (\boldsymbol{vM})_i \geq s_i\right\}\right| = (q - s_i)q^{m-1},$$

and so

$$|S| = \sum_{i=0}^{n-1}(q - s_i)q^{m-1} = nq^m - q^{m-1}\sum_{i=0}^{n-1} s_i. \tag{A.2}$$

The lemma follows from combining (A.1) and (A.2). □

We are now in a position to introduce an alternative *non-constructive* proof for Proposition 4.2.

An alternative for Proposition 4.2. Let $\boldsymbol{s} \in \Sigma$. In order to simplify notation, we assume without loss of generality that $\sum_{i=d_0-2}^{n-1} s_i \leq q-1$. Let $\boldsymbol{w} \in \mathbb{F}_q^n$. We wish to find $\boldsymbol{z} \in \mathbb{F}_q^l$ such that $w_i + (\boldsymbol{zH}_0)_i \geq s_i$ for many indices i.
As the $d_0 - 2$ leftmost columns of $\boldsymbol{H}_0$ are independent, there exists an invertible matrix $\boldsymbol{T} \in \mathbb{F}_q^{l\times l}$ such that

$$\boldsymbol{TH}_0 = \begin{bmatrix} I_{d_0-2} & \boldsymbol{A} \\ \boldsymbol{0} & \boldsymbol{B} \end{bmatrix},$$

where I_{d_0-2} denotes the identity matrix of size $d_0 - 2$.
For $i \in [d_0 - 2]$, we choose $z_i = s_i - w_i$ and write

$$\boldsymbol{v} = \boldsymbol{w} + \boldsymbol{z} \cdot (I_{d_0-2} \mid \boldsymbol{A}).$$

By definition, $v_i = s_i$ for all $i \in [d_0 - 2]$.
As any $d_0 - 1$ columns of $\boldsymbol{TH}_0$ are independent, no column of $\boldsymbol{B}$ consists of only zeroes. Lemma A.1 implies that there is an $\boldsymbol{\eta} \in \mathbb{F}_q^{l-d_0+2}$ such that

$$\left| \left\{ i \in [d_0 - 2, n-1] \;\middle|\; w_i + (\boldsymbol{\eta B})_i \geq s_i \right\} \right| \;\geq\; n - d_0 + 2 - \left\lfloor \frac{1}{q} \sum_{i=d_0-2}^{n-1} s_i \right\rfloor .$$

Combining this with the fact that $v_i = s_i$ for all $i \in [d_0 - 2]$, we infer that for all indices $i \in [n]$, $w_i + ((\boldsymbol{z}, \boldsymbol{\eta})\boldsymbol{TH}_0)_i \geq s_i$. □

A.5 Earlier Findings for Gilbert-Varshamov-like Bound

We provide earlier proof of the GV-type bound for Construction 4.2 and Construction 4.3 in Section 4.5.1, which gives an *uncontrollable* code length n and dimension k, i.e., between two values. These constructions demand a code that includes the all-one vector. In the proof, we show the existence of a code that contains the all-one vector as a codeword. We have presented these outcomes in [APW20].

Theorem A.1 (Gilbert-Varshamov-like bound)**.** *Let the positive integers n, $k \leq n$, $d \leq n$, q fulfill:*

$$\sum_{i=0}^{d-2} \binom{n-1}{i} (q-1)^i < q^{n-k}. \tag{A.3}$$

Then, there exists an $[n', k', d]_q$ code that contains the all-one vector, where n', and k' satisfy:

$$n - d + 2 \leq n' \leq n + 1, \qquad k - d + 2 \leq k' \leq k + 1.$$

The parity-check matrix of this $[n', k', d]_q$ code can be constructed as shown in the proof.

Proof. Similar to the proof of the standard Gilbert–Varshamov bound, we construct a systematic parity-check matrix by adding columns $\boldsymbol{h}_l$ for $l = k+1, k+2, ...$ to a $k \times k$ identity matrix as long as:

$$\sum_{i=0}^{d-2} \binom{l-1}{i} \cdot (q-1)^i < q^{n-k}. \tag{A.4}$$

Recall from the proof of the Gilbert-Varshamov bound that this condition ensures that there exists a column $\boldsymbol{h}_l$ that is linearly independent of any collection of $d-2$ other columns.

If (A.4) is not fulfilled anymore for $l = n+1$, we append an additional parity-check column $\boldsymbol{p}$ to the previous n columns such that the sum of each row is zero (i.e., the weight is even in the binary case). This matrix is therefore:

$$\boldsymbol{H}_e := \left[\ \Big(\underbrace{\boldsymbol{h}_1, \dots, \boldsymbol{h}_n}_{n} \Big) \ \Big| \ \boldsymbol{p} \ \right],$$

where

$$\sum_{i=1}^{n} \boldsymbol{h}_i + \boldsymbol{p} = \boldsymbol{0}. \tag{A.5}$$

However, for $\boldsymbol{H}_e$, we cannot guarantee anymore that any $d-1$ columns are linearly independent (as $\boldsymbol{p}$ might be linearly dependent on a small number of $\boldsymbol{h}_i$'s.). Therefore, in the following, we possibly remove a few columns from $\boldsymbol{H}_e$ to recover this property while still having zero row sums.

If $\boldsymbol{p}$ is linearly independent of any $d-1$ columns in $\boldsymbol{h}_1, \dots, \boldsymbol{h}_n$, we define $\boldsymbol{H} := \boldsymbol{H}_e$.

Else $\boldsymbol{p}$ is linearly dependent of $\Delta \leq d-2$ columns $\{\boldsymbol{h}_{i_1}, \boldsymbol{h}_{i_2}, \dots, \boldsymbol{h}_{i_\Delta}\} \subseteq \{\boldsymbol{h}_1, \boldsymbol{h}_2, \dots, \boldsymbol{h}_n\}$, and $\boldsymbol{p}$ is a linear combination of these Δ columns:

$$\boldsymbol{p} = \sum_{j=1}^{\Delta} \boldsymbol{h}_{i_j} \cdot a_j, \ \text{ where } a_j \in \{1, 2, \dots, q-1\}.$$

Thus with $1 \leq \Delta_1 \leq \dots \leq \Delta_{q-1} \leq \Delta$ (by assuming w.l.o.g. an ordering on the indices),

$$\boldsymbol{p} = \sum_{j=1}^{\Delta_1} \boldsymbol{h}_{i_j} + 2 \cdot \sum_{j=\Delta_1+1}^{\Delta_2} \boldsymbol{h}_{i_j} + \dots + (q-1) \cdot \sum_{j=\Delta_{q-1}+1}^{\Delta} \boldsymbol{h}_{i_j}. \tag{A.6}$$

We can rewrite (A.5) as:

$$\sum_{i=1 \setminus \{i_1, \dots, i_\Delta\}}^{n} \boldsymbol{h}_i + \sum_{j=1}^{\Delta_1} \boldsymbol{h}_{i_j} + \sum_{j=\Delta_1+1}^{\Delta_2} \boldsymbol{h}_{i_j} + \dots + \sum_{j=\Delta_{q-1}+1}^{\Delta} \boldsymbol{h}_{i_j} + \boldsymbol{p} = \boldsymbol{0}.$$

Combining this with (A.6) yields:

$$\begin{aligned} &\sum_{i=1 \setminus \{i_1, \dots, i_\Delta\}}^{n} \boldsymbol{h}_i + \sum_{j=1}^{\Delta_1} \boldsymbol{h}_{i_j} + \sum_{j=\Delta_1+1}^{\Delta_2} \boldsymbol{h}_{i_j} + \dots + \sum_{j=\Delta_{q-1}+1}^{\Delta} \boldsymbol{h}_{i_j} + \boldsymbol{p} \\ &+ \sum_{j=1}^{\Delta_1} \boldsymbol{h}_{i_j} + 2 \cdot \sum_{j=\Delta_1+1}^{\Delta_2} \boldsymbol{h}_{i_j} + \dots + (q-1) \cdot \sum_{j=\Delta_{q-1}+1}^{\Delta} \boldsymbol{h}_{i_j} - \boldsymbol{p}. \\ &= \boldsymbol{0}. \end{aligned}$$

Therefore,

$$\begin{aligned}\sum_{i=1\setminus\{i_1,\dots,i_\Delta\}}^{n} \boldsymbol{h}_i + (2 \mod q)\sum_{j=1}^{\Delta_1} \boldsymbol{h}_{i_j} + (3 \mod q)\sum_{j=\Delta_1+1}^{\Delta_2} \boldsymbol{h}_{i_j} \\ + \dots + (q-1 \mod q)\sum_{j=\Delta_{q-2}+1}^{\Delta_{q-1}} \boldsymbol{h}_{i_j} = \boldsymbol{0}. \end{aligned} \tag{A.7}$$

Therefore, the matrix

$$\boldsymbol{H} := \left(\underbrace{\boldsymbol{h}'_1, \dots, \boldsymbol{h}'_{n-\Delta}}_{n-\Delta} \middle| \underbrace{2\boldsymbol{h}_{i_1}, \dots, 2\boldsymbol{h}_{i_{\Delta_1}}}_{\Delta_1} \middle| \dots \middle| \underbrace{-\boldsymbol{h}_{i_{\Delta_{q-2}+1}}, \dots, -\boldsymbol{h}_{i_{\Delta_{q-1}}}}_{\Delta_{q-1}-\Delta_{q-2}} \right)$$

where $\boldsymbol{h}'_1, \dots, \boldsymbol{h}'_{n-\Delta} = \{\boldsymbol{h}_1, \dots, \boldsymbol{h}_n\} \setminus \{\boldsymbol{h}_{i_1}, \dots, \boldsymbol{h}_{i_\Delta}\}$, has sum equal to zero in all rows due to (A.7) and any $d-1$ columns are linearly independent since they are all columns (times a non-zero scalar) of the matrix $(\boldsymbol{h}_1, \dots, \boldsymbol{h}_n)$.

The number of columns n' of $\boldsymbol{H}$ is bounded by

$$n - d + 2 \leq n - \Delta \leq n' \leq n + 1,$$

where $n' = n+1$ if $\boldsymbol{p}$ was linearly independent of any $d-2$ other columns and therefore no columns have to be removed.

Substituting n in l of (A.4), we obtain:

$$\sum_{i=0}^{d-2} \binom{n-1}{i} (q-1)^i < q^{n-k}. \tag{A.8}$$

Since $n \leq n' + d - 2$ and since $n - k = n' - k'$ (the number of rows did not change), we get

$$\sum_{i=0}^{d-2} \binom{n'+d-3}{i} (q-1)^i < q^{n'-k'}.$$

Thus, if this is true, there exists an $[n', k', d]_q$ code that contains the all-one vector, where $k' = n' - (n' - k') = n' - (n - k) = n' - n + k \geq n - d + 2 - n + k = k - d + 2$. □

Corollary A.1. *Let $u < q$ and let (A.3) hold, i.e., such that an $[n', k', d]_q$ code that contains the all-one vector as codeword exists. Then, there is a $(q-1, 1, \lfloor \frac{d-1}{2} \rfloor)$ PSMC of length n'.*

Proof. In Theorem 4.2 (cf. Section 4.5.1), it was shown that if the all-one vector is a codeword of a code with minimum distance d, then for $u < q$, there is a $(q-1, 1, \lfloor \frac{d-1}{2} \rfloor)$ PSMC. □

Author's Related Publications

[AC22] H. Al Kim and K. J. Chan. "Codes for Preventing Zeros at Partially Defective Memory Positions". In: *2022 IEEE Information Theory Workshop (ITW)*. 2022, pp. 297–302. DOI: 10 . 1109 / ITW54588 . 2022 . 9965909. URL: https://ieeexplore.ieee.org/document/9965909.

[APTW23a] H. Al Kim, S. Puchinger, L. Tolhuizen, and A. Wachter-Zeh. "Coding and bounds for partially defective memory cells". In: *Designs, Codes and Cryptography DCC* (2023).

[APTW23b] H. Al Kim, S. Puchinger, L. Tolhuizen, and A. Wachter-Zeh. "Trading Partially Stuck Cells with Errors". In: Extended Abstract accepted (and nominated as a candidate for "Memorable Award Paper") in the 14th Non-Volatile Memories Workshop (NVMW'23) at the University of California, San Diego. 2023. URL: http://nvmw.ucsd.edu/program/.

[APW19] H. Al Kim, S. Puchinger, and A. Wachter-Zeh. "Error Correction for Partially Stuck Memory Cells". In: *2019 XVI International Symposium "Problems of Redundancy in Information and Control Systems" (REDUNDANCY)*. 2019, pp. 87–92. DOI: 10 . 1109 / REDUNDANCY48165 . 2019 . 9003352. URL: https : / / ieeexplore . ieee . org / document / 9003352.

[APW20] H. Al Kim, S. Puchinger, and A. Wachter-Zeh. "Bounds and Code Constructions for Partially Defect Memory Cells". In: *2020 Algebraic and Combinatorial Coding Theory (ACCT)*. 2020, pp. 1–7. DOI: 10.1109/ACCT51235 . 2020 . 9383410. URL: https : / / ieeexplore . ieee . org/document/9383410.

[APW21] H. Al Kim, S. Puchinger, and A. Wachter-Zeh. "Coding and Bounds for Partially Defective Memory Cells". In: Extended Abstract in the 12th Annual Non-Volatile Memories Workshop (NVMW'21) at the University of California, San Diego. 2021. URL: http : / / nvmw . ucsd . edu / program-2021/.

[YAPW23] Y. Yehezkeally, H. Al Kim, S. Puchinger, and A. Wachter-Zeh. "Bounds on Mixed Codes with Finite Alphabets". In: *the 2023 IEEE Information Theory Workshop (ITW2023), 23-28 April 2023, Saint-Malo, France.* 2023. URL: https://arxiv.org/abs/2212.09314.

Bibliography

[Bas65] L. A. Bassalygo. “New upper bounds for error correcting codes”. In: *Problems Inform. Transmission* 1.4 (1965), 32–35. URL: https://www.mathnet.ru/php/archive.phtml?wshow=paper&jrnid=ppi&paperid=762&option_lang=eng.

[BB13] S. Ball and A. Blokhuis. “A Bound for the Maximum Weight of a Linear Code”. In: *SIAM Journal on Discrete Mathematics* 27.1 (2013), pp. 575–583.

[Ber84] E. R. Berlekamp. *Algebraic Coding Theory.* revised ed. Aegean Park Press, 1984.

[BG04] M Bossert and E Gabidulin. “Polyalphabetic Codes”. In: *Dept. TAIT, University of Ulm, Germany* (2004).

[BHOS98] A. Brouwer, H. Hamalainen, P. Ostergard, and N. Sloane. “Bounds on mixed binary/ternary codes”. In: *IEEE Transactions on Information Theory* 44.1 (1998), pp. 140–161. DOI: 10.1109/18.651001.

[Bla03] R. E. Blahut. *Algebraic Codes for Data Transmission.* 1st ed. Cambridge University Press, 2003.

[Bla+93] I. F. Blake, X. Gao, R. C. Mullin, S. A. Vanstone, and T. Yaghoobian. *Applications of Finite Fields.* 1st ed. Springer, 1993.

[BS77] I Belov and A. M. Shashin. “Codes that correct triple defects in memory”. In: *(in Russian) Problems Inf. Transmiss.,* 13.4 (1977), 62—65.

[Bur+10] G. Burr, M. Breitwisch, M. Franceschini, D. Garetto, K. Gopalakrishnan, B. Jackson, B. Kurdi, C. Lam, L. Lastras, A. Padilla, B. Rajendran, S. Raoux, and R. Shenoy. “Phase change memory technology”. English (US). In: *Journal of Vacuum Science and Technology B:Nanotechnology and Microelectronics* 28.2 (2010), pp. 223–262. ISSN: 2166-2746. DOI: 10.1116/1.3301579.

[BV87] J. Borden and A. Vinck. “On coding for ’stuck-at’ defects (Corresp.)” In: *IEEE Transactions on Information Theory* 33.5 (1987), pp. 729–735. DOI: 10.1109/TIT.1987.1057347.

[Che85] C.-L. Chen. “Linear codes for masking memory defects (Corresp.)” In: *IEEE Transactions on Information Theory* 31.1 (1985), pp. 105–106. DOI: 10.1109/TIT.1985.1056992.

[Che+22] J. Cheriyan, A. Roberts, C. Roberts, M. J. Graves, I. Patterson, R. A. Slough, R. Schroyer, D. Fernando, S. Kumar, S. Lee, G. J. M. Parker, L. Sarov-Blat, C. McEniery, J. Middlemiss, D. Sprecher, and R. L. Janiczek. “Evaluation of dynamic contrast-enhanced MRI measures of lung congestion and endothelial permeability in heart failure: A prospective method validation study”. en. In: *J. Magn. Reson. Imaging* 56.2 (Aug. 2022), pp. 450–461.

[CKCH14] S.-g. Cho, D. Kim, J. Choi, and J. Ha. “Block-Wise Concatenated BCH Codes for NAND Flash Memories”. In: *IEEE Transactions on Communications* 62.4 (2014), pp. 1164–1177. DOI: 10.1109/TCOMM.2014.021514.130287.

[Com+07] C. M. Compagnoni, A. S. Spinelli, R. Gusmeroli, A. L. Lacaita, S. Beltrami, A. Ghetti, and A. Visconti. “First evidence for injection statistics accuracy limitations in NAND Flash constant-current Fowler-Nordheim programming”. In: *2007 IEEE International Electron Devices Meeting*. 2007, pp. 165–168. DOI: 10.1109/IEDM.2007.4418892.

[CS11] J. Choi and K. S. Seol. “3D approaches for non-volatile memory”. In: *2011 Symposium on VLSI Technology - Digest of Technical Papers*. 2011, pp. 178–179.

[CWW14] T.-Y. Chen, A. R. Williamson, and R. D. Wesel. “Increasing flash memory lifetime by dynamic voltage allocation for constant mutual information”. In: *2014 Information Theory and Applications Workshop (ITA)*. 2014, pp. 1–5. DOI: 10.1109/ITA.2014.6804242.

[CZW08] B. Chen, X. Zhang, and Z. Wang. “Error correction for multi-level NAND flash memory using Reed-Solomon codes”. In: *2008 IEEE Workshop on Signal Processing Systems*. 2008, pp. 94–99. DOI: 10.1109/SIPS.2008.4671744.

[DLZ10] G. Dong, S. Li, and T. Zhang. “Using Data Postcompensation and Predistortion to Tolerate Cell-to-Cell Interference in MLC nand Flash Memory”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 57.10 (2010), pp. 2718–2728. DOI: 10.1109/TCSI.2010.2046966.

[Don+12] G. Dong, Y. Pan, N. Xie, C. Varanasi, and T. Zhang. “Estimating Information-Theoretical nand Flash Memory Storage Capacity and its Implication to Memory System Design Space Exploration”. In: *IEEE*

Transactions on Very Large Scale Integration (VLSI) Systems 20.9 (2012), pp. 1705–1714. DOI: 10.1109/TVLSI.2011.2160747.

[DS16] L. Dolecek and F. Sala. *Channel Coding Methods for Non-Volatile Memories.* 2016.

[Dum87] I. I. Dumer. "On linear defect-correcting codes". In: *Proc. 1987 Int. Workshop on Convolutional Codes and Multiuser Communication.* 1987, pp. 222–225.

[Dum89] I. I. Dumer. "Asymptotically Optimal Codes Correcting Memory Defects of Fixed Multiplicity". In: *Problemy Peredachi Informatsii* 25.4 (1989), pp. 3–10.

[Dum90] I. I. Dumer. "Asymptotically Optimal Linear Codes Correcting Defects of Linearly Increasing Multiplicity". In: *Problemy Peredachi Informatsii* 26.2 (1990), pp. 3–17.

[EG93] T. Etzion and G. Greenberg. "Constructions for perfect mixed codes and other covering codes". In: *IEEE Transactions on Information Theory* 39.1 (1993), pp. 209–214. DOI: 10.1109/18.179360.

[Etz22] T. Etzion. *Perfect codes and related structures.* eng. Singapore: World Scientific, 2022. ISBN: 9789811255885.

[For07] B. A. Forouzan. *Data Communications and Networking.* 4th ed. New York: McGraw-Hill Education, 2007.

[FT91] A. Frohlich and M. J. Taylor. *Algebraic Number Theory.* Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1991. DOI: 10.1017/CBO9781139172165.

[GD15] R. Gabrys and L. Dolecek. "Constructions of Nonbinary WOM Codes for Multilevel Flash Memories". In: *IEEE Transactions on Information Theory* 61.4 (2015), pp. 1905–1919. DOI: 10.1109/TIT.2015.2394400.

[GDS12] L. M. Grupp, J. D. Davis, and S. Swanson. "The Bleak Future of NAND Flash Memory". In: *Proceedings of the 10th USENIX Conference on File and Storage Technologies.* FAST'12. San Jose, CA: USENIX Association, 2012, p. 2.

[GHSY12] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin. "On the Locality of Codeword Symbols". In: *IEEE Transactions on Information Theory* 58.11 (2012), pp. 6925–6934. DOI: 10.1109/TIT.2012.2208937.

[GPB09] B Gleixner, F Pellizzer, and R Bez. "Reliability characterization of Phase Change Memory". In: *2009 10th Annual Non-Volatile Memory Technology Symposium (NVMTS).* Portland, OR: IEEE, Oct. 2009.

[Gri60] J. H. Griesmer. "A Bound for Error-Correcting Codes". In: *IBM Journal of Research and Development* 4.5 (1960), pp. 532–542.

[GRS19] V. Guruswami, A. Rudra, and M. Sudan. *Essential Coding Theory.* University at Buffalo, 2019. URL: `https://cse.buffalo.edu/faculty/atri/courses/coding-theory/book/web-coding-book.pdf`.

[GSD14] R. Gabrys, F. Sala, and L. Dolecek. “Coding for Unreliable Flash Memory Cells”. In: *IEEE Communications Letters* 18.9 (2014), pp. 1491–1494. DOI: `10.1109/LCOMM.2014.2344677`.

[Ham50] R Hamming. *Error-Detecting and Error-Correcting Codes,” The Bell Systems Technical.* 1950.

[Hee83] C. Heegard. “Partitioned Linear Block Codes for Computer Memory with’Stuck-at’Defects”. In: *IEEE Transactions on Information Theory* 29.6 (1983), pp. 831–842.

[Hel+14] M. Helm, J.-K. Park, A. Ghalam, J. Guo, C. wan Ha, C. Hu, H. Kim, K. Kavalipurapu, E. Lee, A. Mohammadzadeh, D. Nguyen, V. Patel, T. Pekny, B. Saiki, D. Song, J. Tsai, V. Viajedor, L. Vu, T. Wong, J. H. Yun, R. Ghodsi, A. D’Alessandro, D. Di Cicco, and V. Moschiano. “19.1 A 128Gb MLC NAND-Flash device using 16nm planar cell”. In: *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC).* 2014, pp. 326–327. DOI: `10.1109/ISSCC.2014.6757454`.

[Hol+21] L. Holzbaur, H. Liu, A. Neri, S. Puchinger, J. Rosenkilde, V. Sidorenko, and A. Wachter-Zeh. “Decoding of Interleaved Alternant Codes”. In: *IEEE Transactions on Information Theory* 67.12 (2021), pp. 8016–8033. DOI: `10.1109/TIT.2021.3115432`.

[HP10] W. C. Huffman and V. Pless. *Fundamentals of Error-Correcting Codes.* Cambridge University Press, 2010.

[HPYW21] L. Holzbaur, S. Puchinger, E. Yaakobi, and A. Wachter-Zeh. “Correctable Erasure Patterns in Product Topologies”. In: *2021 IEEE International Symposium on Information Theory (ISIT).* 2021, pp. 2054–2059. DOI: `10.1109/ISIT45174.2021.9518208`.

[HS71] M. Herzog and J. Schonheim. “Linear and nonlinear single-error-correcting perfect mixed codes”. In: *Information and Control* 18.4 (1971), pp. 364–368. ISSN: 0019-9958. DOI: `https://doi.org/10.1016/S0019-9958(71)90464-5`. URL: `https://www.sciencedirect.com/science/article/pii/S0019995871904645`.

[HT72] C. R. P. Hartmann and K. K. Tzeng. “Generalizations of the BCH Bound”. In: *Inf. Control.* 20 (1972), pp. 489–498.

[Hwa+05] S.-S. Hwang, H.-C. Lee, H. W. Ro, D. Y. Yoon, and Y.-C. Joe. "Porosity content dependence of TDDB lifetime and flat band voltage shift by cu diffusion in porous spin-on low-k". In: *2005 IEEE International Reliability Physics Symposium, 2005. Proceedings. 43rd Annual.* San Jose, CA, USA: IEEE, 2005.

[Im+15] J.-W. Im, W.-P. Jeong, D.-H. Kim, S.-W. Nam, D.-K. Shim, M.-H. Choi, H.-J. Yoon, D.-H. Kim, Y.-S. Kim, H.-W. Park, D.-H. Kwak, S.-W. Park, S.-M. Yoon, W.-G. Hahn, J.-H. Ryu, S.-W. Shim, K.-T. Kang, S.-H. Choi, J.-D. Ihm, Y.-S. Min, I.-M. Kim, D.-S. Lee, J.-H. Cho, O.-S. Kwon, J.-S. Lee, M.-S. Kim, S.-H. Joo, J.-H. Jang, S.-W. Hwang, D.-S. Byeon, H.-J. Yang, K.-T. Park, K.-H. Kyung, and J.-H. Choi. "7.2 A 128Gb 3b/cell V-NAND flash memory with 1Gb/s I/O rate". In: *2015 IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers.* 2015, pp. 1–3. DOI: 10.1109/ISSCC.2015.7062960.

[Jia07] A. Jiang. "On The Generalization of Error-Correcting WOM Codes". In: *2007 IEEE International Symposium on Information Theory.* 2007, pp. 1391–1395. DOI: 10.1109/ISIT.2007.4557417.

[JMSB08] A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck. "Rank modulation for flash memories". In: *2008 IEEE International Symposium on Information Theory.* 2008, pp. 1731–1735. DOI: 10.1109/ISIT.2008.4595284.

[Joh63] S. Johnson. "Improved asymptotic bounds for error-correcting codes". In: *IEEE Transactions on Information Theory* 9.3 (1963), pp. 198–205. DOI: 10.1109/TIT.1963.1057841.

[JSB08] A. Jiang, M. Schwartz, and J. Bruck. "Error-correcting codes for rank modulation". In: *2008 IEEE International Symposium on Information Theory.* 2008, pp. 1736–1740. DOI: 10.1109/ISIT.2008.4595285.

[KH14] D. Kim and J. Ha. "Quasi-primitive block-wise concatenated BCH codes for NAND flash memories". In: *2014 IEEE Information Theory Workshop (ITW 2014).* 2014, pp. 611–615. DOI: 10.1109/ITW.2014.6970904.

[KKY78] A Kuznetsov, T Kasami, and S Yamamura. "An error correcting scheme for defective memory". en. In: *IEEE Trans. Inf. Theory* 24.6 (1978), pp. 712–718.

[KL97] V. Krachkovsky and Y. X. Lee. "Decoding for iterative Reed-Solomon coding schemes". In: *IEEE Transactions on Magnetics* 33.5 (1997), pp. 2740–2742. DOI: 10.1109/20.617715.

[KT74] A. Kuznetsov and B. Tsybakov. “Coding for memories with defective cells,” in: *(in Russian) Problems Inf. Transmiss.* Vol. 10. 2. 1974, pp. 52–60.

[Kuz85] A Kuznetsov. *Coding in a channel with generalized defects and random errors.* Vol. 21. 1. 1985, 28—34.

[LCPK03] J.-D. Lee, J.-H. Choi, D. Park, and K. Kim. “Data retention characteristics of sub-100 nm NAND flash memory cells”. In: *IEEE Electron Device Letters* 24.12 (2003), pp. 748–750. DOI: 10.1109/LED.2003.820645.

[Lev98] V. Levenshtein. “Universal bounds for codes and designs, in Handbook of Coding Theory”. In: (1998), pp. 1–149. URL: https://keldysh.ru/papers/1998/prep_vw.asp?pid=2319&lg=e.

[Lid+97] R. Lidl, H. Niederreiter, P. Cohn, G. Rota, and B. Doran. *Finite Fields.* EBL-Schweitzer v. 20, pt. 1. Cambridge University Press, 1997. ISBN: 9780521392310. URL: https://books.google.de/books?id=xqMqxQTFUkMC.

[Lin92] J. H. van Lint. *Introduction to coding theory.* en. 2nd ed. Graduate texts in mathematics. Berlin, Germany: Springer, July 1992.

[LKD78] V. V. Losev, V. K. Konopel’ko, and Y. D. Daryakin. “Double-and-triple-defect-correcting”. In: *(in Russian) Problems Inf. Transmiss.,* 14.4 (1978), 98—101.

[Loe97] H.-A. Loeliger. “Averaging bounds for lattices and linear codes”. In: *IEEE Transactions on Information Theory* 43.6 (1997), pp. 1767–1773. DOI: 10.1109/18.641543.

[LW01] J. H. van Lint and R. M. Wilson. *A Course in Combinatorics.* Cambridge, U.K.; New York: Cambridge University Press, 2001. ISBN: 9780511674877 0511674872 9780511671623 0511671628 9780511987045 0511987048.

[MBZ13] A. Mazumdar, A. Barg, and G. Zemor. “Constructions of Rank Modulation Codes”. In: *IEEE Transactions on Information Theory* 59.2 (2013), pp. 1018–1029. DOI: 10.1109/TIT.2012.2221121.

[MC+09] C. Monzio Compagnoni, M. Ghidotti, A. L. Lacaita, A. S. Spinelli, and A. Visconti. “Random Telegraph Noise Effect on the Programmed Threshold-Voltage Distribution of Flash Memories”. In: *IEEE Electron Device Letters* 30.9 (2009), pp. 984–986. DOI: 10.1109/LED.2009.2026658.

[Mie^{+}04] N. Mielke, H. Belgal, I. Kalastirsky, P. Kalavade, A. Kurtz, Q. Meng, N. Righos, and J. Wu. “Flash EEPROM threshold instabilities due to charge trapping during program/erase cycling”. In: *IEEE Transactions on Device and Materials Reliability* 4.3 (2004), pp. 335–344. DOI: 10.1109/TDMR.2004.836721.

[Mie^{+}06] N. Mielke, H. P. Belgal, A. Fazio, Q. Meng, and N. Righos. “Recovery Effects in the Distributed Cycling of Flash Memories”. In: *2006 IEEE International Reliability Physics Symposium Proceedings*. 2006, pp. 29–35. DOI: 10.1109/RELPHY.2006.251188.

[MK09] Y. Maeda and H. Kaneko. “Error Control Coding for Multilevel Cell Flash Memories Using Nonbinary Low-Density Parity-Check Codes”. In: *2009 24th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*. 2009, pp. 367–375. DOI: 10.1109/DFT.2009.25.

[MK90] J. Metzner and E. Kapturowski. “A general decoding technique applicable to replicated file disagreement location and concatenated code decoding”. In: *IEEE Transactions on Information Theory* 36.4 (1990), pp. 911–917. DOI: 10.1109/18.53757.

[MRRW77] R. McEliece, E. Rodemich, H. Rumsey, and L. Welch. “New upper bounds on the rate of a code via the Delsarte-MacWilliams inequalities”. In: *IEEE Transactions on Information Theory* 23.2 (1977), pp. 157–166. DOI: 10.1109/TIT.1977.1055688.

[MS77] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error Correcting Codes*. Vol. 16. Elsevier/North-Holland, 1977.

[NS09] M. Navon and A. Samorodnitsky. “Linear programming bounds for codes via a covering argument”. en. In: *Discrete Comput. Geom.* 41.2 (Mar. 2009), pp. 199–207.

[ORS86] P. OLIVO, B. RICCO, and E. SANGIORGI. “High-field-induced voltage-dependent oxide charge”. English. In: *Applied physics letters* (1986). ISSN: 0003-6951.

[PBK14] K.-T. Park, D.-S. Byeon, and D.-H. Kim. “A world’s first product of three-dimensional vertical NAND Flash memory and beyond”. In: *2014 14th Annual Non-Volatile Memory Technology Symposium (NVMTS)*. 2014, pp. 1–5. DOI: 10.1109/NVMTS.2014.7060840.

[Pir^{+}04] A Pirovano, A Redaelli, F Pellizzer, F Ottogalli, M Tosi, D Ielmini, A. L. Lacaita, and R Bez. “Reliability study of phase-change nonvolatile memories”. en. In: *IEEE trans. device mater. reliab.* 4.3 (Sept. 2004), pp. 422–427.

[PPMP14] T. Parnell, N. Papandreou, T. Mittelholzer, and H. Pozidis. “Modelling of the threshold voltage distributions of sub-20nm NAND flash memory”. In: *2014 IEEE Global Communications Conference*. 2014, pp. 2351–2356. DOI: 10.1109/GLOCOM.2014.7037159.

[PSS06] S. Perkins, A. Sakhnovich, and D. Smith. “On an upper bound for mixed error-correcting codes”. In: *IEEE Transactions on Information Theory* 52.2 (2006), pp. 708–712. DOI: 10.1109/TIT.2005.862107.

[QJS13] M. Qin, A. A. Jiang, and P. H. Siegel. “Parallel programming of rank modulation”. In: *2013 IEEE International Symposium on Information Theory*. 2013, pp. 719–723. DOI: 10.1109/ISIT.2013.6620320.

[Roo79] C. Roos. “On the structure of convolutional and cyclic convolutional codes”. In: *IEEE Transactions on Information Theory* 25.6 (1979), pp. 676–683. DOI: 10.1109/TIT.1979.1056108.

[Rot06] R. M. Roth. *Introduction to Coding Theory*. Cambridge University Press, 2006. DOI: 10.1017/CBO9780511808968.005.

[RPW21] J. Renner, S. Puchinger, and A. Wachter-Zeh. “Decoding High-Order Interleaved Rank-Metric Codes”. In: *2021 IEEE International Symposium on Information Theory (ISIT)*. 2021, pp. 19–24. DOI: 10.1109/ISIT45174.2021.9518085.

[RS82] R. L. Rivest and A. Shamir. “How to reuse a "write-once" memory”. In: *Information and Control* 55.1 (1982), pp. 1–19. ISSN: 0019-9958. DOI: https://doi.org/10.1016/S0019-9958(82)90344-8. URL: https://www.sciencedirect.com/science/article/pii/S0019995882903448.

[SC19a] A. Solomon and Y. Cassuto. “Error-correcting WOM codes: Concatenation and joint design”. In: *IEEE Trans. Inf. Theory* 65.9 (Sept. 2019), pp. 5529–5546.

[SC19b] A. Solomon and Y. Cassuto. “Error-Correcting WOM Codes: Concatenation and Joint Design”. In: *IEEE Transactions on Information Theory* 65.9 (2019), pp. 5529–5546. DOI: 10.1109/TIT.2019.2917519.

[SH22] I. Shomorony and R. Heckel. “Information-Theoretic Foundations of DNA Data Storage”. In: *Foundations and Trends® in Communications and Information Theory* 19.1 (2022), pp. 1–106. ISSN: 1567-2190. DOI: 10.1561/0100000117. URL: http://dx.doi.org/10.1561/0100000117.

[Sha48] C. E. Shannon. *A Mathematical Theory of Communication,” The Bell Systems Technical*. 1948.

[Sid+05] V. Sidorenko, G. Schmidt, E. Gabidulin, M. Bossert, and V. Afanassiev. "On polyalphabetic block codes". In: *IEEE Information Theory Workshop, 2005.* 2005, 4 pp.–. DOI: 10.1109/ITW.2005.1531889.

[Sol74] G. Solomon. "A Note on Alphabet Codes and Fields of Computation". In: *Inf. Control.* 25 (1974), pp. 395–398.

[Sta22] Statista. *Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2020, with forecasts from 2021 to 2025.* Accessed on 2022-10-24. 2022. URL: https://www.statista.com/statistics/871513/worldwide-data-created/#:~:text=The%20total%20amount%20of%20data,to%20more%20than%20180%20zettabytes (visited on 10/24/2022).

[TGKO75] B. S. Tsybakov, S. I. Gelfand, A. V. Kuznetsov, and S. I. Ortyukov. "Reliable computation and reliable storage of information". In: *Proc. IEEE-USSR Workshop* (Dec. 1975).

[Tsy75a] B. S. Tsybakov. "Defects and error correction". In: *(in Russian) Problems Inf. Transmiss.,* 11.1 (1975), 21—30.

[Tsy75b] B. S. Tsybakov. "Group additive defect-correcting codes". In: *(in Russian) Problems Inf. Transmiss* 11.1 (1975), 111—113.

[Tsy77] B. S. Tsybakov. "Bounds for the codes correcting errors and defects". In: *Problemy Peredachi Informatsii* 13.2 (1977), pp. 11–22.

[TTN96] K. Takeuchi, T. Tanaka, and H. Nakamura. "A double-level-V/sub th/ select gate array architecture for multilevel NAND flash memories". In: *IEEE Journal of Solid-State Circuits* 31.4 (1996), pp. 602–609. DOI: 10.1109/4.499738.

[Wan+14] J. Wang, K. Vakilinia, T.-Y. Chen, T. Courtade, G. Dong, T. Zhang, H. Shankar, and R. Wesel. "Enhanced Precision Through Multiple Reads for LDPC Decoding in Flash Memories". In: *IEEE Journal on Selected Areas in Communications* 32.5 (2014), pp. 880–891. DOI: 10.1109/JSAC.2014.140508.

[WCW15] H. Wang, T.-Y. Chen, and R. D. Wesel. "Histogram-based Flash channel estimation". In: *2015 IEEE International Conference on Communications (ICC).* 2015, pp. 283–288. DOI: 10.1109/ICC.2015.7248335.

[WSS15] A. Wachter-Zeh, M. Stinner, and V. Sidorenko. "Convolutional Codes in Rank Metric With Application to Random Network Coding". In: *IEEE Transactions on Information Theory* 61.6 (2015), pp. 3199–3213. DOI: 10.1109/TIT.2015.2424930.

[WWCW16] H. Wang, N. Wong, T.-Y. Chen, and R. D. Wesel. “Using Dynamic Allocation of Write Voltage to Extend Flash Memory Lifetime”. In: *IEEE Transactions on Communications* 64.11 (2016), 4474–4486. ISSN: 0090-6778. DOI: `10.1109/tcomm.2016.2607707`. URL: `http://dx.doi.org/10.1109/TCOMM.2016.2607707`.

[WY16] A. Wachter-Zeh and E. Yaakobi. “Codes for Partially Stuck-at Memory Cells”. In: *IEEE Transactions on Information Theory* 62.2 (2016), pp. 639–654.

[Yaa+12] E. Yaakobi, S. Kayser, P. H. Siegel, A. Vardy, and J. K. Wolf. “Codes for Write-Once Memories”. In: *IEEE Transactions on Information Theory* 58.9 (2012), pp. 5985–5999. DOI: `10.1109/TIT.2012.2200291`.

[Yan+06] H. Yang, H. Kim, S.-i. Park, J. Kim, S.-h. Lee, J.-k. Choi, D. Hwang, C. Kim, M. Park, K.-h. Lee, Y.-k. Park, J. K. Shin, and J.-t. Kong. “Reliability Issues and Models of sub-90nm NAND Flash Memory Cells”. In: *2006 8th International Conference on Solid-State and Integrated Circuit Technology Proceedings*. 2006, pp. 760–762. DOI: `10.1109/ICSICT.2006.306478`.

[Zha+13] K. Zhao, W. Zhao, H. Sun, T. Zhang, X. Zhang, and N. Zheng. “LDPC-in-SSD: Making Advanced Error Correction Codes Work Effectively in Solid State Drives”. In: *Proceedings of the 11th USENIX Conference on File and Storage Technologies*. FAST’13. San Jose, CA: USENIX Association, 2013, 243–256.

[ZPJ10] F. Zhang, H. D. Pfister, and A. Jiang. “LDPC codes for rank modulation in flash memories”. In: *2010 IEEE International Symposium on Information Theory*. 2010, pp. 859–863. DOI: `10.1109/ISIT.2010.5513603`.

[ZW14] M. Zwolenski and L. Weatherill. “The digital universe: Rich data and the increasing value of the internet of things”. In: *Journal of Telecommunications and the Digital Economy* 2.3 (2014), pp. 47–1.