Rewriting Schemes for Flash Memory

Thesis by

Eyal En Gad

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

2015

(Defended February 25, 2015)

To my mother

and

my wife Anna

Acknowledgments

I would like to start by acknowledging my advisor Jehoshua (Shuki) Bruck, for his enormous contributions to this thesis and to my professional and personal development during my time at Caltech. Shuki's advices about problem selection, motivation, solutions, and presentation led me to enjoy my work greatly and to take pride in my results. He also taught me how to be always optimistic and kind, and to enjoy fruitful collaborations. I feel very lucky and grateful for having the opportunity to study under him.

During my graduate studies I was lucky to collaborate and learn from many other wonderful researchers. Moshe Schwartz introduced me to the beauty and utility of combinatorial coding theory, and taught me many useful ways to present research ideas. His sharp mind and kind attitude were very inspiring for me. Michael Langberg taught me many concepts in combinatorics and information theory with a very thorough approach. I also learned a lot from him about how to explain research results in an interesting and clear way, and how to always be kind and pleasant with everyone. Andrew Jiang introduced me to wonderful ideas about coding in flash memory. His energy and excitement are contagious and make every discussion with him very joyful. Joerg Kliewer introduced me to many concepts in modern coding theory, and is always exited to explore new research directions. The discussions with him gave me many exiting ideas and new perspectives.

I also had the opportunity to collaborate with several of my lab-mates at the Paradise lab, which was a wonderful experience. I had a great time working with Eitan Yaakobi during the two years he spent in the Paradise lab. He brought his endless optimism and energy and his valuable experience in flash memory coding. Yue Li is a great collaborator and friend since he joined the group. I am thankful for the practical perspective he gave

me towards research, and for his constant smile that kept the atmosphere in lab always cheerful. I am also thankful for many challenging racquetball matches with him. Another great collaborator and lab-mate is Wentao Huang, with whom I worked in the last part of my studies. I am thankful to Wentao's dedication and innovative thinking that made this work very exiting.

I am also thankful to the many other lab members that I did not have the chance to collaborate with: Daniel Wilhelm, Hongchao Zhou, Zhiying Wang, David Lee, Itzhak Tamo, Lulu Qian, Farzad Farnoud, and Siddharth Jain.

I would like to thank Robert Mateescu, who hosted me in an internship at HGST. I had a great time during the three months I spent at the Storage Architecture group. I am very grateful to my collaborators at HGST - Filip Blagojevic, Cyril Guyot and Zvonimir Bandic - and to the other friendly people at the research group, including Dejan Vucinic, Qingbo Wang, Luiz Franca-Neto, and Jorge Campello.

I would like to thank my thesis committee: Michelle Effros, Babak Hassibi and P. P. Vaidyanathan. Their illuminating questions and comments provided me with different points of view and new ways of thinking.

I am very thankful for the remote support of my family and friends in Israel. My mother Orna, my stepfather Uri, my brother Oded and my sister Neta. Your love and support was an indispensable ingredient of the success and joy I had in my doctoral studies.

Finally, I was lucky to meet my wonderful and beautiful wife Anna during my time at Caltech. This was the most successful part of my PhD effort. I am very grateful for her love and partnership, which gave a new meaning to my work. I am also thankful for my parents-in-law Manuel and Suzette and my sibling-in-law Sergio and Priscilla for making me part of their family.

Abstract

Flash memory is a leading storage media with excellent features such as random access and high storage density. However, it also faces significant reliability and endurance challenges. In flash memory, the charge level in the cells can be easily increased, but removing charge requires an expensive erasure operation. In this thesis we study rewriting schemes that enable the data stored in a set of cells to be rewritten by only increasing the charge level in the cells. We consider two types of modulation scheme; a convectional modulation based on the absolute levels of the cells, and a recently-proposed scheme based on the relative cell levels, called rank modulation. The contributions of this thesis to the study of rewriting schemes for rank modulation include the following: we

- propose a new method of rewriting in rank modulation, beyond the previously proposed method of "push-to-the-top";
- study the limits of rewriting with the newly proposed method, and derive a tight upper bound of 1 bit per cell;
- extend the rank-modulation scheme to support rankings with repetitions, in order to improve the storage density;
- derive a tight upper bound of 2 bits per cell for rewriting in rank modulation with repetitions;
- construct an efficient rewriting scheme that asymptotically approaches the upper bound of 2 bit per cell.

The next part of this thesis studies rewriting schemes for a conventional absolute-levels modulation. The considered model is called "write-once memory" (WOM). We focus on WOM schemes that achieve the capacity of the model. In recent years several capacity-achieving WOM schemes were proposed, based on polar codes and randomness extractors. The contributions of this thesis to the study of WOM scheme include the following: we

- propose a new capacity-achieving WOM scheme based on sparse-graph codes, and show its attractive properties for practical implementation;
- improve the design of polar WOM schemes to remove the reliance on shared randomness and include an error-correction capability.

The last part of the thesis studies the local rank-modulation (LRM) scheme, in which a sliding window going over a sequence of real-valued variables induces a sequence of permutations. The LRM scheme is used to simulate a single conventional multi-level flash cell. The simulated cell is realized by a Gray code traversing all the relative-value states where, physically, the transition between two adjacent states in the Gray code is achieved by using a single "push-to-the-top" operation. The main results of the last part of the thesis are two constructions of Gray codes with asymptotically-optimal rate.

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Rewriting in Flash Memory

This thesis deals with coding schemes for data storage in flash memory. Flash memory is a leading storage media with many excellent features such as random access and high storage density. However, it also faces significant reliability and endurance challenges. Flash memory contains floating gate cells. The cells are electrically charged with electrons and can represent multiple levels according to the number of electrons they contain. The most conspicuous property of flash-storage technology is its inherent asymmetry between cell programming and cell erasing. While it is fast and simple to increase a cell level, reducing its level requires a long and cumbersome operation of first erasing its entire containing block ($\sim 10^6$cells) and only then programming the cells [8]. Such block erasures are not only time consuming, but also degrade the lifetime of the memory. A typical block can generally tolerate at most $10^4 - 10^5$ erasures.

To reduce the amount of block erasures, the focus of this thesis is on schemes for the *rewriting* of data in the memory. Rewriting schemes allow to update information stored in a set of flash cells solely by increasing the cell levels (without decreasing the level of any cell). Rewriting schemes were in fact studied before the emergence of flash memory, as memories whose cells transit irreversibly between states have been common since the beginning of the data storage technology. Examples include punch cards and digital optical discs, where a cell can change from a 0-state to a 1-state but not vice versa. In addition, non-volatile memories other than flash memory also exhibit an asymmetric writing property, such as phase-change

Table 1.1: WOM-Code Example

Data bits	First write	Second write (if data changes)
00	000	111
10	100	011
01	010	101
11	001	110

memories.

The first example of a rewriting scheme was given by Rivest and Shamir in 1982 [69]. This example considers a rewriting model called write-once memory (WOM), in which the memory cells take binary values, and can only change for 0 to 1. The example is a simple WOM code that enables the recording of two bits of information in three memory cells twice. The encoding and decoding rules for this WOM-code are described in a tabular form in Table 1.1. It is easy to verify that after the first 2-bit data vector is encoded into a 3-bit codeword, if the second 2-bit data vector is different from the first, the 3-bit codeword into which it is encoded does not change any code bit 1 into a code bit 0, ensuring that it can be recorded in the write-once medium.

WOM codes with more cells and more data bits were studied extensively in recent years, since the emergence of the flash memory application. In this thesis we make several contribution to the study of WOM codes. In addition to WOM, the thesis also studies rewriting in a different data-representation scheme, called rank modulation. We turn our attention now to describe the rewriting setting in rank modulation.

1.1 Rank Modulation

Rank modulation was recently proposed [45]. In this scheme a set of n memory cells represents information according to the *ranking* of the cell levels. For example, we can use a set of 3 cells, labeled from 1 to 3, such that each cell has a distinct charge level. We then rank the cells according to their charge levels, and obtain one of $3! = 6$ possible permutations over the set $\{1, 2, 3\}$. A possible ranking would be, for example, cell 3 with the highest level, then

cell 1 and then cell 2 with the lowest level. Each ranking can represent a distinct information message, and so the 3 cells in this example store together $\log_2 6$ bits. It was suggested in [45] that the rank-modulation scheme speeds up data writing by eliminating the over-shooting problem in flash memories. In addition, it also increases the data retention by mitigating the effect of charge leakage.

Rank-modulation rewriting codes were proposed in [45, Section IV], with respect to a rewriting method called "push-to-the-top". In this rewriting method, the charge level of a single cell is pushed up to be higher than that of any other cell in the ranking. In other words, a push-to-the-top operation changes the rank of a single cell to be the highest. A rewriting operation involves a sequence of push-to-the-top operations that transforms the cell ranking to represent a desired updated data. The cells, however, have an upper limit on their possible charge levels. Therefore, after a certain number of rewriting operations, the user must resort to the expensive erasure operation in order to continue updating the memory. The concept of rewriting codes was proposed in order to control the trade-off between the number of data updates and the amount of data stored in each update. Note that the number of performed push-to-the-top operations determines when an expensive block erasure is required. However, the number of rewriting operations itself does not affect the triggering of the block erasure. Therefore, rewriting operations that require fewer push-to-the-top operations can be seen as *cheaper*, and are therefore more desirable. Nevertheless, limiting the memory to cheap rewriting operations would reduce the number of potential rankings to write, and therefore would reduce the amount of information that could be stored. We refer to the number of push-to-the-top operations in a given rewriting operation as the *cost of rewriting*. The study in [45, Section IV] considers rewriting codes with a constrained rewriting cost.

We study rank-modulation in Part II of the thesis. The first contribution of this part is a modification of the framework of rank-modulation rewriting codes, in two ways. First, we modify the rank-modulation scheme to allow rankings with *repetitions*, meaning that multiple cells can share the same rank, where the number of cells in each rank is predetermined. And second, we extend the rewriting operation to allow pushing a cell's level above that of any

desired cell, instead of only above the level of the top cell. We justify both modifications and devise an appropriate notion of rewriting cost. Specifically, we define the cost to be the difference between the charge level of the *highest* cell, *after* the writing operation, to the charge level of the highest cell *before* the rewriting operation. We suggest and explain why the new cost function compares fairly to that of the push-to-the-top model. We then go on to study rewriting codes in the modified framework.

We measure the storage rate of rewriting codes by the ratio between the number of stored information bits in each write, to the number of cells in the ranking. We study the case in which the number of cells is large (and asymptotically growing), while the cost constraint is a *constant*, as this case appears to be fairly relevant for practical applications. In the model of push-to-the-top rewriting which was studied in [45, Section IV], the storage rate vanishes when the number of cells grows. Our first interesting result is that the asymptotic storage rate in our modified framework converges into a *positive* value (that depends on the cost constraint). Specifically, using rankings *without* repetitions, i.e. the original rank modulation scheme with the modified rewriting operation, and the minimal cost constraint of a single unit, the best storage rate converges to a value of 1 bit per cell. Moreover, when ranking with repetitions is allowed, the best storage rate with a minimal cost constraint converges to a value of 2 bits per cell.

Motivated by these positive results, we design an explicit construction of rank-modulation rewriting codes, together with computationally efficient encoding and decoding algorithms. The main ingredients in the code construction are write-once memory (WOM) schemes. We focus on ranking with repetitions, where both the number of cells in each rank and the number of ranks are growing. In this case, we show how to make use of capacity-achieving WOM codes to construct rank-modulation rewriting codes with an asymptotically optimal rate for any given cost constraint.

1.2 Write-Once Memory

In Part III of this thesis we propose two new constructions of WOM codes. These constructions can be used with an absolute-level modulation, or with the rank-modulation rewriting schemes proposed in Part II.

The constructions proposed in this thesis are based on sparse-graph codes and on polar codes. Sparse-graph WOM codes are introduced in this thesis, while polar WOM codes were discovered earlier (in [7]), and this thesis contributes to their study. Many other types of WOM codes were proposed in the literature, including [21, 25, 43, 44, 86, 88, 91].

An important feature of both WOM schemes proposed in this thesis is that they achieve the capacity of the WOM model (given in [36]). The sparse-graph WOM codes achieve the capacity of the WOM for two writes, while the polar WOM codes achieve the capacity for any number of writes. In both schemes we also propose methods to integrate the WOM codes with error-correcting codes.

1.2.1 Sparse-Graph WOM codes

The sparse-graph WOM codes are proposed for the case of two writes. The construction is based on binary erasure quantization with low-density-generator-matrix (LDGM) codes. The encoding is performed by the iterative quantization studied by Martinian and Yedidia [58], which is a message-passing algorithm similar to the decoding of low-density-parity-check (LDPC) codes. As LDPC codes have been widely adopted by commercial flash memory controllers, the hardware architectures of message-passing algorithms have been well understood and highly optimized in practice. Therefore, the proposed codes are implementation-friendly for practitioners. Extensive simulations show that the rewriting performance of the scheme compares favorably with that of the polar WOM code in the rate region where a low rewriting failure rate is desired. For instance, we show that our code allows user to write 40% more information by rewriting with very high success probability. We note that the iterative quantization algorithm of [58] was used in [10] in a different way for the problem of

information embedding, which shares some similarity with our model.

Moreover, the proposed construction is extended with error correction. We use conjugate code pairs studied in the context of quantum error correction [35]. As an example, we construct LDGM WOM codes whose codewords also belong to BCH codes. Therefore, our codes allow to use any decoding algorithm of BCH codes. The latter have been implemented in most commercial flash memory controllers. We also present two additional approaches to add error correction, and compare their performance.

1.2.2 Polar WOM codes

Polar WOM codes were introduced in [7]. The construction is based on polar lossy source codes of Korada and Urbanke [50], which are based on the polar channel codes of Arikan [1]. We propose a different construction of polar WOM codes, which offers two main advantages over the previously proposed codes. First, our codes do not require shared randomness between the encoder and decoder, which is required in the original polar WOM codes. And second, we extend the codes with error correction capability.

The proposed polar codes are analyzed with respect to the model of channel coding with state information available at the encoder, proposed by Gelfand and Pinsker [16, page 178] [28]. This model can be seen as a generalization of the model of point-to-point channel coding model. This implies that the proposed coding scheme can be used also for point-to-point channel coding. In this setting our schemes provides an additional contribution, since it possesses favorable properties compared with known capacity-achieving schemes for asymmetric point-to-point channels.

We consider two noisy WOM models, and provide two variations of the polar WOM scheme. We show that each of these variations achieves the capacity of the respected noise model.

1.3 Local Rank Modulation

In Part IV we consider a different approach to rewriting in the rank-modulation scheme. The considered rewriting approach, described first in [45], is the following: a set of n cells, over which the rank-modulation scheme is applied (without repetitions), is used to simulate a single conventional multi-level flash cell with $n!$ levels corresponding to the alphabet $\{0, 1, \ldots, n! - 1\}$. The simulated cell supports an operation which raises its value by 1 modulo $n!$. This is the only required operation in many rewriting schemes for flash memories (see [42, 43, 87]), and it is realized in [45] by a Gray code traversing the $n!$ states where, physically, the transition between two adjacent states in the Gray code is achieved by using a single "push-to-the-top" operation.

Most generally, a gray code is a sequence of distinct elements from an ambient space such that adjacent elements in the sequence are "similar". Ever since their original publication by Gray [33], the use of Gray codes has reached a wide variety of areas, such as storage and retrieval applications [11], processor allocation [12], statistics [14], hashing [23], puzzles [27], ordering documents [53], signal encoding [54], data compression [68], circuit testing [70], and more.

The Gray code was first introduced as a sequence of distinct binary vectors of fixed length, where every adjacent pair differs in a single coordinate [33]. It has since been generalized to sequences of distinct states $s_1, s_2, \ldots, s_k \in S$ such that for every $i < k$ there exists a function in a predetermined set of transitions $\tau \in T$ such that $s_{i+1} = \tau(s_i)$ (see [77] for an excellent survey). In the context of rank modulation in [45], the state space consisted of permutations over n elements, and the "push-to-the-top" operations were the allowed transitions. This operation was studied since it is a simple programming operation that is quick and eliminates the over-programming problem. We also note that generating permutations using "push-to-the-top" operations is of independent interest, called "nested cycling" in [78] (see also references therein), motivated by a fast "push-to-the-top" operation (cycling) available on some computer architectures.

A drawback to the rank-modulation scheme (without repetitions) is the need for a large

number of comparisons when reading the induced permutation from a set of n cell-charge levels. Instead, in a recent work [85], the n cells are locally viewed through a sliding window, resulting in a sequence of small permutations that require less comparisons. We call this the *local rank-modulation scheme*. The aim of this part of the thesis is to study Gray codes for the local rank-modulation scheme. Another generalization of Gray codes for rank modulation is the "snake-in-the-box" codes in [89].

We present two constructions of asymptotically rate-optimal Gray codes. The first construction considers the case of $(1, 2, n)$-LRMGC, while the second construction considers the more general case of (s, t, n)-LRMGC. However, the first construction also considers an additional property, in which the codes have a constant weight.

Part II

Rewriting with Rank Modulation

Chapter 2

Model and Limits of Rewriting Schemes

The material in Chapters 2 and 3 was presented in part in [17, 18, 22].

2.1 Modifications to the Rank-Modulation Scheme

In this section we motivate and define the rank-modulation scheme, together with the proposed modification to the scheme and to the rewriting process.

2.1.1 Motivation for Rank Modulation

The rank-modulation scheme is motivated by the physical and architectural properties of flash memories (and similar non-volatile memories). First, the charge injection in flash memories is a noisy process in which an overshooting may occur. When the cells represent data by their *absolute* value, such overshooting results in a different stored data than the desired one. And since the cell level cannot be decreased, the charge injection is typically performed iteratively and therefore slowly, to avoid such errors. However, in rank modulation such overshooting errors can be corrected without decreasing the cell levels, by pushing other cells to form the desired ranking. An additional issue in flash memories is the leakage of charge from the cells over time, which introduces additional errors. In rank modulation, such leakage is significantly less problematic, since it behaves similarly in spatially close cells, and

thus is not likely to change the cells' ranking. A hardware implementation of the scheme was recently designed on flash memories [47].

We note that the motivation above is valid also for the case of ranking with repetitions, which was not considered in previous literature with respect to the rank-modulation scheme. We also note that the rank-modulation scheme in some sense reduces the amount of information that can be stored, since it limits the possible state that the cells can take. For example, it is not allowed for all the cell levels to be the same. However, this disadvantage might be worth taking for the benefits of rank modulation, and this is the case in which we are interested in this part of the thesis.

2.1.2 Representing Data by Rankings with Repetitions

In this subsection we extend the rank-modulation scheme to allow rankings with repetitions, and formally define the extended demodulation process. We refer to rankings with repetitions as *permutations of multisets*, where rankings without repetitions are permutations of sets. Let $M = \{a_1^{z_1}, \ldots, a_q^{z_q}\}$ be a multiset of q distinct elements, where each element a_i appears z_i times. The positive integer z_i is called the multiplicity of the element a_i, and the cardinality of the multiset is $n = \sum_{i=1}^{q} z_i$. For a positive integer n, the set $\{1, 2, \ldots, n\}$ is labeled by $[n]$. We think of a permutation σ of the multiset M as a partition of the set $[n]$ into q disjoint subsets, $\sigma = (\sigma(1), \sigma(2), \ldots, \sigma(q))$, such that $|\sigma(i)| = z_i$ for each $i \in [q]$, and $\cup_{i\in[q]}\sigma(i) = [n]$. We also define the inverse permutation σ^{-1} such that for each $i \in [q]$ and $j \in [n]$, $\sigma^{-1}(j) = i$ if j is a member of the subset $\sigma(i)$. We label σ^{-1} as the length-n vector $\sigma^{-1} = (\sigma^{-1}(1), \sigma^{-1}(2), \ldots, \sigma^{-1}(n))$. For example, if $M = \{1, 1, 2, 2\}$ and $\sigma = (\{1, 3\}, \{2, 4\})$, then $\sigma^{-1} = (1, 2, 1, 2)$. We refer to both σ and σ^{-1} as a permutation, since they represent the same object.

Let $\mathfrak{S}_M$ be the set of all permutations of the multiset M. By abuse of notation, we view $\mathfrak{S}_M$ also as the set of the inverse permutations of the multiset M. For a given cardinality n and number of elements q, it is easy to show that the number of multiset permutations is maximized if the multiplicities of all of the elements are equal. Therefore, to simplify the pre-

sentation, we take most of the multisets in this thesis to be of the form $M = \{1^z, 2^z, \ldots, q^z\}$, and label the set $\mathfrak{S}_M$ by $\mathfrak{S}_{q,z}$.

Consider a set of n memory cells, and denote $\boldsymbol{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ as the *cell-state vector*. The values of the cells represent voltage levels, but we do not pay attention to the units of these values (i.e., Volt). We represent information on the cells according to the mutiset permutation that their values induce. This permutation is derived by a demodulation process.

Demodulation: Given positive integers q and z, a cell-state vector $\boldsymbol{x}$ of length $n = qz$ is demodulated into a permutation $\pi_{\boldsymbol{x}}^{-1} = (\pi_{\boldsymbol{x}}^{-1}(1), \pi_{\boldsymbol{x}}^{-1}(2), \ldots, \pi_{\boldsymbol{x}}^{-1}(n))$. Note that while $\pi_{\boldsymbol{x}}^{-1}$ is a function of q, z and $\boldsymbol{x}$, q and z are not specified in the notation since they will be clear from the context. The demodulation is performed as follows: first, let $k_1, \ldots, k_n$ be an order of the cells such that $x_{k_1} \leq x_{k_2} \leq \cdots \leq x_{k_n}$. Then, for each $j \in [n]$, assign $\pi_{\boldsymbol{x}}^{-1}(k_j) = \lceil j/z \rceil$.

Example 2.1: *Let $q = 3$, $z = 2$ and so $n = qz = 6$. Assume that we wish to demodulate the cell-state vector $\boldsymbol{x} = (1, 1.5, 0.3, 0.5, 2, 0.3)$. We first order the cells according to their values: $(k_1, k_2, \ldots, k_6) = (3, 6, 4, 1, 2, 5)$, since the third and sixth cells have the smallest value, and so on. Then we assign*

$$\pi_{\boldsymbol{x}}^{-1}(k_1 = 3) = \lceil 1/2 \rceil = 1,$$

$$\pi_{\boldsymbol{x}}^{-1}(k_2 = 6) = \lceil 2/2 \rceil = 1,$$

$$\pi_{\boldsymbol{x}}^{-1}(k_3 = 4) = \lceil 3/2 \rceil = 2,$$

and so on, and get the permutation $\pi_{\boldsymbol{x}}^{-1} = (2, 3, 1, 2, 3, 1)$. Note that $\pi_{\boldsymbol{x}}^{-1}$ is in $\mathfrak{S}_{3,2}$.

Note that $\pi_{\boldsymbol{x}}$ is not unique if for some $i \in [q]$, $x_{k_{zi}} = x_{k_{zi+1}}$. In this case, we define $\pi_{\boldsymbol{x}}$ to be illegal and denote $\pi_{\boldsymbol{x}} = F$. We label Q_M as the set of all cell-state vectors that demodulate into a valid permutation of M. That is, $Q_M = \{\boldsymbol{x} \in \mathbb{R}^n \mid \pi_{\boldsymbol{x}} \neq F\}$. So for all $\boldsymbol{x} \in Q_M$ and $i \in [q]$, we have $x_{k_{zi}} < x_{k_{zi+1}}$. For $j \in [n]$, the value $\pi^{-1}(j)$ is called the *rank* of cell j in the permutation π.

2.1.3 Rewriting in Rank Modulation

In this subsection we extend the rewriting operation in the rank-modulation scheme. Previous work considered a writing operation called "push-to-the-top", in which a certain cell is pushed to be the highest in the ranking [45]. Here we suggest to allow to push a cell to be higher than the level of *any* specific other cell. We note that this operation is still resilient to overshooting errors, and therefore benefits from the advantage of fast writing, as the push-to-the-top operations.

We model the flash memory such that when a user wishes to store a message on the memory, the cell levels can only increase. When the cells reach their maximal levels, an expensive erasure operation is required. Therefore, in order to maximize the number of writes between erasures, it is desirable to raise the cell levels as little as possible on each write. For a cell-state vector $\boldsymbol{x} \in Q_M$, denote by $\Gamma_{\boldsymbol{x}}(i)$ the highest level among the cells with rank i in $\pi_{\boldsymbol{x}}$. That is,

$$\Gamma_{\boldsymbol{x}}(i) = \max_{j \in \pi_{\boldsymbol{x}}(i)} \{x_j\}.$$

Let $\boldsymbol{s}$ be the cell-state vector of the memory before the writing process takes place, and let $\boldsymbol{x}$ be the cell-state vector after the write. In order to reduce the possibility of error in the demodulation process, a certain gap must be placed between the levels of cells with different ranks. Since the cell levels's units are somewhat arbitrary, we set this gap to be the value 1, for convenience. The following modulation method minimizes the increase in the cell levels.

Modulation: Writing a permutation π on a memory with state $\boldsymbol{s}$. The output is the new memory state, denoted by $\boldsymbol{x}$.

1. For each $j \in \pi(1)$, assign $x_j \Leftarrow s_j$.

2. For $i = 2, 3, \ldots, q$, for each $j \in \pi(i)$, assign

$$x_j \Leftarrow \max\{s_j, \Gamma_{\boldsymbol{x}}(i-1) + 1\}.$$

Example 2.2: *Let $q = 3$, $z = 2$ and so $n = qz = 6$. Let the state be $\boldsymbol{s} = (2.7, 4, 1.5, 2.5, 3.8, 0.5)$ and the target permutation be $\pi^{-1} = (1, 1, 2, 2, 3, 3)$. In step 1 of the modulation process, we notice that $\pi(1) = \{1, 2\}$, and so we set*

$$x_1 \Leftarrow s_1 = 2.7$$

and

$$x_2 \Leftarrow s_2 = 4.$$

In step 2 we have $\pi(2) = \{3, 4\}$ and $\Gamma_{\boldsymbol{x}}(1) = \max\{x_1, x_2\} = \max\{2.7, 4\} = 4$, so we set

$$x_3 \Leftarrow \max\{s_3, \Gamma_{\boldsymbol{x}}(1) + 1\} = \max\{1.5, 5\} = 5$$

and

$$x_4 \Leftarrow \max\{s_4, \Gamma_{\boldsymbol{x}}(1) + 1\} = \max\{2.5, 5\} = 5.$$

And in the last step we have $\pi(3) = \{5, 6\}$ and $\Gamma_{\boldsymbol{x}}(2) = 5$, so we set

$$x_5 \Leftarrow \max\{3.8, 6\} = 6$$

and

$$x_6 \Leftarrow \max\{0.5, 6\} = 6.$$

In summary, we get $\boldsymbol{x} = (2.7, 4, 5, 5, 6, 6)$, which demodulates into $\pi_{\boldsymbol{x}}^{-1} = (1, 1, 2, 2, 3, 3) = \pi^{-1}$, as required.

Since the cell levels cannot decrease, we must have $x_j \geq s_j$ for each $j \in [n]$. In addition, for each j_1 and j_2 in $[n]$ for which $\pi^{-1}(j_1) > \pi^{-1}(j_2)$, we must have $x_{j_1} > x_{j_2}$. Therefore, the proposed modulation process minimizes the increase in the levels of all the cells.

2.2 Definition and Limits of Rank-Modulation Rewriting Codes

Remember that the level x_j of each cell is upper bounded by a certain value. Therefore, given a state $\boldsymbol{s}$, certain permutations π might require a block erasure before writing, while others might not. In addition, some permutations might get the memory state *closer* to a state in which an erasure is required than other permutations. In order to maximize the number of writes between block erasures, we add redundancy by letting *multiple* permutations represent the *same* information message. This way, when a user wishes to store a certain message, she could choose one of the permutations that represent the required message such that the chosen permutation will increase the cell levels in the least amount. Such a method can increase the longevity of the memory in the expense of the amount of information stored on each write. The mapping between the permutations and the messages they represent is called a rewriting code.

To analyze and design rewriting codes, we focus on the difference between $\Gamma_{\boldsymbol{x}}(q)$ and $\Gamma_{\boldsymbol{s}}(q)$. Using the modulation process we defined above, the vector $\boldsymbol{x}$ is a function of $\boldsymbol{s}$ and π, and therefore the difference $\Gamma_{\boldsymbol{x}}(q) - \Gamma_{\boldsymbol{s}}(q)$ is also a function of $\boldsymbol{s}$ and π. We label this difference by $\alpha(\boldsymbol{s} \to \pi) = \Gamma_{\boldsymbol{x}}(q) - \Gamma_{\boldsymbol{s}}(q)$ and call it the *rewriting cost*, or simply the cost. We motivate this choice by the following example. Assume that the difference between the maximum level of the cells and $\Gamma_{\boldsymbol{s}}(q)$ is 10 levels. Then only the permutations π which satisfy $\alpha(\boldsymbol{s} \to \pi) \leq 10$ can be written to the memory without erasure. Alternatively, if we use a rewriting code that guarantees that for any state $\boldsymbol{s}$, any message can be stored with, say, cost no greater than 1, then we can guarantee to write 10 more times to the memory before an erasure will be required. Such rewriting codes are the focus of this part of the thesis.

The cost $\alpha(\boldsymbol{s} \to \pi)$ is defined according to the vectors $\boldsymbol{s}$ and $\boldsymbol{x}$. However, it will be helpful for the study of rewriting codes to have some understanding of the cost in terms of the demodulation of the state $\boldsymbol{s}$ and the permutation π. To establish such connection, we

assume that the state $\boldsymbol{s}$ is a result of a previous modulation process. This assumption is reasonable, since we are interested in the scenario of multiple successive rewriting operations. In this case, for each $i \in [q-1]$, $\Gamma_{\boldsymbol{s}}(i+1) - \Gamma_{\boldsymbol{s}}(i) \geq 1$, by the modulation process. Let $\sigma_{\boldsymbol{s}}$ be the permutation obtained from the demodulation of the state $\boldsymbol{s}$. We present the connection in the following proposition.

Proposition 2.1: *Let M be a multiset of cardinality n. If $\Gamma_{\boldsymbol{s}}(i+1) - \Gamma_{\boldsymbol{s}}(i) \geq 1$ for all $i \in [q-1]$, and π is in $\mathfrak{S}_M$, then*

$$\alpha(\boldsymbol{s} \to \pi) \leq \max_{j \in [n]} \{\sigma_{\boldsymbol{s}}^{-1}(j) - \pi^{-1}(j)\} \tag{2.1}$$

with equality if $\Gamma_q(\boldsymbol{s}) - \Gamma_1(\boldsymbol{s}) = q - 1$.

The proof of Proposition 2.1 is brought in Section 2.3. We would take a worst-case approach, and opt to design codes that guarantee that on each rewriting, the value $\max_{j \in [n]} \{\sigma_{\boldsymbol{s}}^{-1}(j) - \pi^{-1}(j)\}$ is bounded. For permutations σ and π in $\mathfrak{S}_{q,z}$, the rewriting cost $\alpha(\sigma \to \pi)$ is defined as

$$\alpha(\sigma \to \pi) = \max_{j \in [n]} \{\sigma^{-1}(j) - \pi^{-1}(j)\}. \tag{2.2}$$

This expression is an asymmetric version of the Chebyshev distance (also known as the L_∞ distance). For simplicity, we assume that the channel is noiseless and don't consider the error-correction capability of the codes. However, such consideration would be essential for practical applications.

2.2.1 Definition of Rank-Modulation Rewriting Codes

A rank-modulation rewriting code is a partition of the set of multiset permutations, such that each part represents a different information message, and each message can be written on each state with a cost that is bounded by some parameter r. A formal definition follows.

Definition 2.1: ***(Rank-modulation rewriting codes)*** *Let q, z, r, and K_R be positive integers, and let $\mathcal{C}$ be a subset of $\mathfrak{S}_{q,z}$ called the codebook. Then a surjective function D_R :*

$\mathcal{C} \to [K_R]$ is a (q, z, K_R, r) ***rank-modulation rewriting code (RM rewriting code)*** *if for each message $m \in [K_R]$ and state $\sigma \in \mathcal{C}$, there exists a permutation π in $D_R^{-1}(m) \subseteq \mathcal{C}$ such that $\alpha(\sigma \to \pi) \leq r$.*

$D_R^{-1}(m)$ is the set of permutations that represent the message m. It could also be insightful to study rewriting codes according to an average cost constraint, assuming some distribution on the source and/or the state. However, we use the wort-case constraint since it is easier to analyze. The amount of information stored with a (q, z, K_R, r) RM rewriting code is $\log K_R$ bits (all of the logarithms in this thesis are binary). Since it is possible to store up to $\log |\mathfrak{S}_{q,z}|$ bits with permutations of a multiset $\{1^z, \ldots, q^z\}$, it could be natural to define the code rate as:

$$R' = \frac{\log K_R}{\log |\mathfrak{S}_{q,z}|}.$$

However, this definition doesn't give much engineering insight into the amount of information stored in a set of memory cells. Therefore, we define the rate of the code as the amount of information stored per memory cell:

$$R = \frac{\log K_R}{qz}.$$

An encoding function E_R for a code D_R maps each pair of message m and state σ into a permutation π such that $D_R(\pi) = m$ and $\alpha(\sigma \to \pi) \leq r$. By abuse of notation, let the symbols E_R and D_R represent both the functions and the algorithms that compute those functions. If D_R is a RM rewriting code and E_R is its associated encoding function, we call the pair (E_R, D_R) a rank-modulation rewrite coding scheme.

Rank-modulation rewriting codes were proposed by Jiang *et al.* in [45], in a more restrictive model than the one we defined above. The model in [45] is more restrictive in two senses. First, the mentioned model used the rank-modulation scheme with permutations of sets only, while here we also consider permutations of multisets. And second, the rewriting operation in the mentioned model was composed only of a cell programming operation called "push-to-the-top", while here we allow a more opportunistic programming approach.

A push-to-the-top operation raises the charge level of a single cell above the rest of the cells in the set. As described above, the model of this chapter allows one to raise a cell level above a subset of the rest of the cells. The rate of RM rewriting codes with push-to-the-top operations and cost of $r = 1$ tends to zero with the increase in the block length n. On the contrary, we will show that the rate of RM rewriting codes with cost $r = 1$ and the model of this chapter tends to 1 bit per cell with permutations of sets, and 2 bits per cell with permutations of multisets.

2.2.2 Limits of Rank-Modulation Rewriting Codes

For the purpose of studying the limits of RM rewriting codes, we define the ball of radius r around a permutation σ in $\mathfrak{S}_{q,z}$ by

$$B_{q,z,r}(\sigma) = \{\pi \in \mathfrak{S}_{q,z} | \alpha(\sigma \to \pi) \leq r\},$$

and derive its size in the following lemma.

Lemma 2.1: *For positive integers q and z, if σ is in $\mathfrak{S}_{q,z}$ then*

$$|B_{q,z,r}(\sigma)| = \binom{(r+1)z}{z}^{q-r} \prod_{i=1}^{r} \binom{iz}{z}.$$

PROOF: Let $\pi \in B_{q,z,r}(\sigma)$. By the definition of $B_{q,z,r}(\sigma)$, for any $j \in \pi(1)$, $\sigma^{-1}(j)-1 \leq r$, and thus $\sigma^{-1}(j) \leq r+1$. Therefore, there are $\binom{(r+1)z}{z}$ possibilities for the set $\pi(1)$ of cardinality z. Similarly, for any $i \in \pi(2)$, $\sigma(i)^{-1} \leq r+2$. So for each fixed set $\pi(1)$, there are $\binom{(r+1)z}{z}$ possibilities for $\pi(2)$, and in total $\binom{(r+1)z}{z}^2$ possibilities for the pair of sets $(\pi(1), \pi(2))$. The same argument follows for all $i \in [q-r]$, so there are $\binom{(r+1)z}{z}^{q-r}$ possibilities for the sets $(\pi(1), \ldots, \pi(q-r))$. The rest of the sets of π: $\pi(q-r+1), \pi(q-r+2), \ldots, \pi(q)$, can take any permutation of the multiset $\{(q-r+1)^z, (q-r+2)^z, \ldots, q^z\}$, giving the statement of the lemma. ∎

Note that the size of $B_{q,z,r}(\sigma)$ is actually *not* a function of σ. Therefore we denote it by $|B_{q,z,r}|$.

Proposition 2.2: *Let D_R be a (q, z, K_R, r) RM rewriting code. Then*

$$K_R \leq |B_{q,z,r}|.$$

PROOF: Fix a state $\sigma \in \mathcal{C}$. By Definition 2.1 of RM rewriting codes, for any message $m \in [K_R]$ there exists a permutation π such that $D_R(\pi) = m$ and π is in $B_{q,z,r}(\sigma)$. It follows that $B_{q,z,r}(\sigma)$ must contain K_R different permutations, and so its size must be at least K_R.■

Corollary 2.1: *Let $R(r)$ be the rate of an (q, z, K_R, r)-RM rewriting code. Then*

$$R(r) < (r+1)H\left(\frac{1}{r+1}\right),$$

where $H(p) = -p\log p - (1-p)\log(1-p)$. In particular, $R(1) < 2$.

PROOF:

$$\begin{aligned}
\log|B_{q,z,r}| =& \sum_{i=1}^{r} \log\binom{iz}{z} + (q-r)\log\binom{(r+1)z}{z} \\
<& r\log\binom{(r+1)z}{z} + (q-r)\log\binom{(r+1)z}{z} \\
=& q\log\binom{(r+1)z}{z} \\
<& q\cdot(r+1)zH\left(\frac{1}{r+1}\right),
\end{aligned}$$

where the last inequality follows from Stirling's formula. So we have

$$R(r) = \frac{\log K_R}{qz} \leq \frac{\log|B_{q,z,r}|}{qz} < (r+1)H\left(\frac{1}{r+1}\right).$$

The case of $r = 1$ follows immediately. ■

We will later show that this bound is in fact tight, and therefore we call it the *capacity* of RM rewriting codes and denote it as

$$C_R(r) = (r+1)H\left(\frac{1}{r+1}\right).$$

Henceforth we omit the radius r from the capacity notation and denote it by C_R. To further motivate the use of multiset permutations rather than set permutation, we can observe the following corollary.

Corollary 2.2: *Let $R(r)$ be the rate of an $(q, 1, K_R, r)$-RM rewriting code. Then $R(r) < \log(r+1)$, and in particular, $R(1) < 1$.*

PROOF: Note first that $|B_{q,z,r}| = (r+1)^{q-r} r!$. So we have

$$\begin{aligned}\log|B_{q,z,r}| &= \log r! + (q-r)\log(r+1)\\ &< r\log(r+1) + (q-r)\log(r+1)\\ &= q\log(r+1).\end{aligned}$$

Therefore,

$$R(r) \leq \frac{\log|B_{q,z,r}|}{q} < \log(r+1),$$

and the case of $r = 1$ follows immediately. ∎

In the case of $r = 1$, codes with multiset permutations could approach a rate close to 2 bits per cell, while there are no codes with set permutations and rate greater than 1 bit per cell. The constructions we present in the next chapter are analyzed only for the case of multiset permutations with a large value of z. We now define two properties that we would like to have in a family of RM rewrite coding schemes. First, we would like the rate of the codes to approach the upper bound of Corollary 2.1. We call this property *capacity achieving*.

Definition 2.2: ***(Capacity-achieving family of RM rewriting codes)*** *For a positive integer i, let the positive integers q_i, z_i, and K_i be some functions of i, and let $n_i = q_i z_i$ and $R_i = (1/n_i) \log K_i$. Then an infinite family of (q_i, z_i, K_i, r) RM rewriting codes is called* ***capacity achieving*** *if*

$$\lim_{i \to \infty} R_i = C_R.$$

The second desired property is computational efficiency. We say that a family of RM rewrite coding schemes $(E_{R,i}, D_{R,i})$ is *efficient* if the algorithms $E_{R,i}$ and $D_{R,i}$ run in polynomial time in $n_i = q_i z_i$. The main result of the next chapter is a construction of an efficient capacity-achieving family of RM rewrite coding schemes.

2.3 Proof of the Cost Function

Proof (of Proposition 2.1): We want to prove that if $\Gamma_{\boldsymbol{s}}(i+1) - \Gamma_{\boldsymbol{s}}(i) \geq 1$ for all $i \in [q-1]$, and π is in $\mathfrak{S}_M$, then

$$\alpha(\boldsymbol{s} \to \pi) \leq \max_{j \in [n]} \{\sigma_{\boldsymbol{s}}^{-1}(j) - \pi^{-1}(j)\}$$

with equality if $\Gamma_{\boldsymbol{s}}(q) - \Gamma_{\boldsymbol{s}}(1) = q - 1$.

The assumption implies that

$$\Gamma_{\boldsymbol{s}}(i) \leq \Gamma_{\boldsymbol{s}}(q) + i - q \tag{2.3}$$

for all $i \in [q]$, with equality if $\Gamma_{\boldsymbol{s}}(q) - \Gamma_{\boldsymbol{s}}(1) = q - 1$.

Next, define a set $U_{i_1,i_2}(\sigma_{\boldsymbol{s}})$ to be the union of the sets $\{\sigma_{\boldsymbol{s}}(i)\}_{i \in [i_1:i_2]}$, and remember that the writing process sets $x_j = s_j$ if $\pi^{-1}(j) = 1$, and otherwise

$$x_j = \max\{s_j, \Gamma_{\boldsymbol{x}}(\pi^{-1}(j) - 1) + 1\}.$$

Now we claim by induction on $i \in [q]$ that

$$\Gamma_{\boldsymbol{x}}(i) \leq i + \Gamma_{\boldsymbol{s}}(q) - q + \max_{j \in U_{1,i}(\pi)} \{\sigma_{\boldsymbol{s}}^{-1}(j) - \pi^{-1}(j)\}. \tag{2.4}$$

In the base case, $i = 1$, and

$$\begin{aligned}
\Gamma_{\boldsymbol{x}}(1) &\overset{(a)}{=} \max_{j \in \pi(1)} \{x_j\} \overset{(b)}{=} \max_{j \in \pi(1)} \{s_j\} \overset{(c)}{\leq} \max_{j \in \pi(1)} \{\Gamma_{\boldsymbol{s}}(\sigma_{\boldsymbol{s}}^{-1}(j))\} \overset{(d)}{\leq} \max_{j \in \pi(1)} \{\Gamma_{\boldsymbol{s}}(q) - q + \sigma_{\boldsymbol{s}}^{-1}(j)\} \\
&\overset{(e)}{=} \Gamma_{\boldsymbol{s}}(q) - q + \max_{j \in \pi(1)} \{\sigma_{\boldsymbol{s}}^{-1}(j) + (1 - \pi^{-1}(j))\} \\
&\overset{(f)}{=} 1 + \Gamma_{\boldsymbol{s}}(q) - q + \max_{j \in U_{1,i}(\pi)} \{\sigma_{\boldsymbol{s}}^{-1}(j) - \pi^{-1}(j)\}
\end{aligned}$$

Where (a) follows from the definition of $\Gamma_{\boldsymbol{x}}(1)$, (b) follows from the modulation process, (c) follows since $\Gamma_{\boldsymbol{s}}(\sigma_{\boldsymbol{s}}^{-1}(j)) = \max_{j' \in \sigma_{\boldsymbol{s}}(\sigma_{\boldsymbol{s}}^{-1}(j))}\{s_{j'}\}$, and therefore $\Gamma_{\boldsymbol{s}}(\sigma_{\boldsymbol{s}}^{-1}(j)) \geq s_j$ for all $j \in [n]$, (d) follows from Equation 2.3, (e) follows since $j \in \pi(1)$, and therefore $\pi^{-1}(j) = 1$, and (f) is just a rewriting of the terms. Note that the condition $\Gamma_{\boldsymbol{s}}(q) - \Gamma_{\boldsymbol{s}}(1) = q - 1$ implies that $s_j = \Gamma_{\boldsymbol{s}}(\sigma_{\boldsymbol{s}}^{-1}(j))$ and $\Gamma_{\boldsymbol{s}}(i) = \Gamma_{\boldsymbol{s}}(q) + i - q$, and therefore equality in (c) and (d).

For the inductive step, we have

$$\begin{aligned}
\Gamma_{\boldsymbol{x}}(i) &\overset{(a)}{=} \max_{j \in \pi(i)} \{x_j\} \\
&\overset{(b)}{=} \max_{j \in \pi(i)} \{\max\{s_j, \Gamma_{\boldsymbol{x}}(i-1) + 1\}\} \\
&\overset{(c)}{\leq} \max\{\max_{j \in \pi(i)} \{s_j\}, (i-1) + \Gamma_{\boldsymbol{s}}(q) - q + \max_{j \in U_{1,i-1}(\pi)} \{\sigma_{\boldsymbol{s}}^{-1}(j) - \pi^{-1}(j)\} + 1\} \\
&\overset{(d)}{\leq} \max\{\max_{j \in \pi(i)} \{\Gamma_{\boldsymbol{s}}(\sigma_{\boldsymbol{s}}^{-1}(j))\}, i + \Gamma_{\boldsymbol{s}}(q) - q + \max_{j \in U_{1,i-1}(\pi)} \{\sigma_{\boldsymbol{s}}^{-1}(j) - \pi^{-1}(j)\}\} \\
&\overset{(e)}{\leq} \max\{\max_{j \in \pi(i)} \{\Gamma_{\boldsymbol{s}}(q) - q + \sigma_{\boldsymbol{s}}^{-1}(j)\}, i + \Gamma_{\boldsymbol{s}}(q) - q + \max_{j \in U_{1,i-1}(\pi)} \{\sigma_{\boldsymbol{s}}^{-1}(j) - \pi^{-1}(j)\}\} \\
&\overset{(f)}{=} \Gamma_{\boldsymbol{s}}(q) - q + \max\{\max_{j \in \pi(i)} \{\sigma_{\boldsymbol{s}}^{-1}(j) + (i - \pi^{-1}(j))\}, i + \max_{j \in U_{1,i-1}(\pi)} \{\sigma_{\boldsymbol{s}}^{-1}(j) - \pi^{-1}(j)\}\} \\
&\overset{(g)}{=} i + \Gamma_{\boldsymbol{s}}(q) - q + \max\{\max_{j \in \pi(i)} \{\sigma_{\boldsymbol{s}}^{-1}(j) - \pi^{-1}(j)\}, \max_{j \in U_{1,i-1}(\pi)} \{\sigma_{\boldsymbol{s}}^{-1}(j) - \pi^{-1}(j)\}\}
\end{aligned}$$

$$\overset{\text{(h)}}{=} i + \Gamma_{\boldsymbol{s}}(q) - q + \max_{j \in U_{1,i}(\pi)} \{\sigma_{\boldsymbol{s}}^{-1}(j) - \pi^{-1}(j)\}$$

Where (a) follows from the definition of $\Gamma_{\boldsymbol{x}}(i)$, (b) follows from the modulation process, (c) follows from the induction hypothesis, (d) follows from the definition of $\Gamma_{\boldsymbol{s}}(\sigma_{\boldsymbol{s}}^{-1}(j))$, (e) follows from Equation 2.3, (f) follows since $\pi^{-1}(j) = i$, and (g) and (h) are just rearrangements of the terms. This completes the proof of the induction claim. As in the base case, we see that if $\Gamma_{\boldsymbol{s}}(q) - \Gamma_{\boldsymbol{s}}(1) = q - 1$ then the inequality in Equation 2.4 becomes an equality.

Finally, taking $i = q$ in Equation 2.4 gives

$$\Gamma_{\boldsymbol{x}}(q) \leq q + \Gamma_{\boldsymbol{s}}(q) - q + \max_{j \in U_{1,q}(\pi)} \{\sigma_{\boldsymbol{s}}^{-1}(j) - \pi^{-1}(j)\} = \Gamma_{\boldsymbol{s}}(q) + \max_{j \in [n]} \{\sigma_{\boldsymbol{s}}^{-1}(j) - \pi^{-1}(j)\}$$

with equality if $\Gamma_{\boldsymbol{s}}(q) - \Gamma_{\boldsymbol{s}}(1) = q - 1$, which completes the proof of the proposition, since $\alpha(\boldsymbol{s} \to \pi)$ was defined as $\Gamma_{\boldsymbol{x}}(q) - \Gamma_{\boldsymbol{s}}(q)$. ∎

Chapter 3

Efficient Rewriting Schemes

3.1 High-Level Construction

The proposed construction is composed of two layers. The higher layer of the construction is described in this section, and two alternative implementations of the lower layer are described in the following two sections. The high-level construction involves several concepts, which we introduce one by one. The first concept is to divide the message into $q - r$ parts, and to encode and decode each part separately. The codes that are used for the different message parts are called "ingredient codes". We demonstrate this concept in Subsection 3.1.1 by an example in which $q = 3$,$z = 2$ and $r = 1$, and the RM code is divided into $q - r = 2$ ingredient codes.

The second concept involves the implementation of the ingredient codes when the parameter z is greater than 2. We show that in this case the construction problem reduces to the construction of the so-called "constant-weight WOM codes". We demonstrate this in Subsection 3.1.2 with a construction for general values of z, where we show that capacity-achieving constant-weight WOM codes lead to capacity achieving RM rewriting codes. Next, in Subsections 3.1.3 and 3.1.4, we generalize the parameters q and r, where these generalizations are conceptually simpler.

Once the construction is general for q, z, and r, we modify it slightly in Subsection 3.1.5 to accommodate a weaker notion of WOM codes, which are easier to construct. The next

two sections present two implementations of capacity-achieving weak WOM codes that can be used to construct capacity-achieving RM rewriting codes.

A few additional definitions are needed for the description of the construction. First, let $2^{[n]}$ denote the set of all subsets of $[n]$. Next, let the function $\theta_n : 2^{[n]} \to \{0,1\}^n$ be defined such that for a subset $S \subseteq [n]$, $\theta_n(S) = (\theta_{n,1}, \theta_{n,2}, \ldots, \theta_{n,n})$ is its characteristic vector, where

$$\theta_{n,j} = \begin{cases} 0 & \text{if } j \notin S \\ 1 & \text{if } j \in S. \end{cases}$$

For a vector $\boldsymbol{x}$ of length n and a subset $S \subseteq [n]$, we denote by $\boldsymbol{x}_S$ the vector of length $|S|$ which is obtained by "throwing away" all the positions of $\boldsymbol{x}$ outside of S. For positive integers $n_1 \leq n_2$, the set $\{n_1, n_1+1, \ldots, n_2\}$ is labeled by $[n_1 : n_2]$. Finally, for a permutation $\sigma \in \mathfrak{S}_{q,z}$, we define the set $U_{i_1,i_2}(\sigma)$ as the union of the sets $\{\sigma(i)\}_{i \in [i_1:i_2]}$ if $i_1 \leq i_2$. If $i_1 > i_2$, we define $U_{i_1,i_2}(\sigma)$ to be the empty set.

3.1.1 A Construction for $q = 3$, $z = 2$ and $r = 1$

In this construction we introduce the concept of dividing the code into multiple ingredient codes. The motivation for this concept comes from a view of the encoding process as a sequence of choices. Given a message m and a state permutation σ, the encoding process needs to find a permutation π that represents m, such that the cost $\alpha(\sigma \to \pi)$ is no greater then the cost constraint r. The cost function $\alpha(\sigma \to \pi)$ is defined in Equation 2.2 as the maximal *drop* in rank among the cells, when moving from σ to π. In other words, we look for the cell that dropped the most amount of ranks from σ to π, and the cost is the number of ranks that this cell has dropped. If cell j is at rank 3 in σ and its rank is changed to 1 in π, it dropped 2 ranks. In our example, since $q = 3$, a drop of 2 ranks is the biggest possible drop, and therefore, if at least one cell dropped by 2 ranks, the rewriting cost would be 2.

In the setting of $q = 3$ ranks, $z = 2$ cells per rank, and cost constraint of $r = 1$, to make sure that a the rewriting cost would not exceed 1, it is enough to ensure that the 2 cells

of rank 3 in σ do not drop into rank 1 in π. So the cells that take rank 1 in π must come from ranks 1 or 2 in σ. This motivates us to look at the encoding process as a sequence of 2 decisions. First, the encoder chooses two cells out of the 4 cells in ranks 1 and 2 in σ to occupy rank 1 in π. Next, after the $\pi(1)$ (the set of cells with rank 1 in π) is selected, the encoder completes the encoding process by choosing a way to arrange the remaining 4 cells in ranks 2 and 3 of π. There are $\binom{4}{2} = 6$ such arrangements, and they all satisfy the cost constraint, since a drop from a rank no greater than 3 into a rank no smaller than 2 cannot exceed a magnitude of 1 rank. So the encoding process is split into two decisions, which define it entirely.

The main concept in this subsection is to think of the message as a pair $\boldsymbol{m} = (m_1, m_2)$, such that the first step of the encoding process encodes m_1, and the second step encodes m_2. The first message part, m_1, is encoded by the set $\pi(1)$. To satisfy the cost constraint of $r = 1$, the set $\pi(1)$ must be chosen from the 4 cells in ranks 1 and 2 in σ. These 4 cells are denoted by $U_{1,2}(\sigma)$. For each m_1 and set $U_{1,2}(\sigma)$, the encoder needs to find 2 cells from $U_{1,2}(\sigma)$ that represent m_1. Therefore, there must be *multiple* selections of 2 cells that represent m_1.

The encoding function for m_1 is denoted by $E_W(m_1, U_{1,2}(\sigma))$, and the corresponding decoding function is denoted by $D_W(\pi(1))$. We denote by $D_W^{-1}(m_1)$ the *set* of subsets that D_W maps into m_1. We denote the number of possible values that m_1 can take by K_W. To demonstrate the code D_W for m_1, we show an example that contains $K_W = 5$ messages.

Example 3.1: *Consider the following code D_W, defined by the values of D_W^{-1}:*

$$
\begin{aligned}
D_W^{-1}(1) &= \Big\{\{1,2\},\{3,4\},\{5,6\}\Big\} \\
D_W^{-1}(2) &= \Big\{\{1,3\},\{2,6\},\{4,5\}\Big\} \\
D_W^{-1}(3) &= \Big\{\{1,4\},\{2,5\},\{3,6\}\Big\} \\
D_W^{-1}(4) &= \Big\{\{1,5\},\{2,3\},\{4,6\}\Big\} \\
D_W^{-1}(5) &= \Big\{\{1,6\},\{2,4\},\{3,5\}\Big\}.
\end{aligned}
$$

To understand the code, assume that $m_1 = 3$ *and* $\sigma^{-1} = (1, 2, 1, 3, 2, 3)$*, so that the cells of ranks 1 and 2 in* σ *are* $U_{1,2}(\sigma) = \{1, 2, 3, 5\}$*. The encoder needs to find a set in* $D_W^{-1}(3)$ *that is a subset of* $U_{1,2}(\sigma) = \{1, 2, 3, 5\}$*. In this case, the only such set is* $\{2, 5\}$*. So the encoder chooses cells 2 and 5 to occupy rank 1 of* π*, meaning that the rank of cells 2 and 5 in* π *is 1, or that* $\pi(1) = \{2, 5\}$*. To find the value of* m_1*, the decoder calculates the function* $D_W(\pi(1)) = 3$*. It is not hard to see that for any values of* m_1 *and* $U_{1,2}(\sigma)$ *(that contains 4 cells), the encoder can find 2 cells from* $U_{1,2}(\sigma)$ *that represent* m_1*.*

The code for m_2 is simpler to design. The encoder and decoder both know the identity of the 4 cells in ranks 2 and 3 of π, so each arrangement of these two ranks can correspond to a different message part m_2. We denote the number of messages in the code for m_2 by K_M, and define the multiset $M = \{2, 2, 3, 3\}$. We also denote the pair of sets $(\pi(2), \pi(3))$ by $\pi_{[2:3]}$. Each arrangement of $\pi_{[2:3]}$ corresponds to a different permutation of M, and encodes a different message part m_2. So we let

$$K_M = |\mathfrak{S}_M| = \binom{4}{2} = 6.$$

For simplicity, we encode m_2 according to the *lexicographic* order of the permutations of M. For example, $m_2 = 1$ is encoded by the permutation $(2, 2, 3, 3)$, $m_2 = 2$ is encoded by $(2, 3, 2, 3)$, and so on. If, for example, the cells in ranks 2 and 3 of π are $\{1, 3, 4, 6\}$, and the message part is $m_2 = 2$, the encoder sets

$$\pi_{[2:3]} = (\pi(2), \pi(3)) = (\{1, 4\}, \{3, 6\}).$$

The bijective mapping form $[K_M]$ to the permutations of M is denoted by $h_M(m_2)$, and the inverse mapping by $h^{-1}(\pi_{[2:3]})$. The code h_M is called an enumerative code.

The message parts m_1 and m_2 are encoded sequentially, but can be decoded in parallel. The number of messages that the RM rewriting code in this example can store is

$$K_R = K_W \times K_M = 5 \times 6 = 30,$$

as each rank stores information independently.

Construction 3.1: *Let* $K_W = 5, q = 3, z = 2, r = 1$*, let* $n = qz = 6$ *and let* (E_W, D_W) *be defined according to Example* 3.1*. Define the multiset* $M = \{2, 2, 3, 3\}$ *and let* $K_M = |\mathfrak{S}_M| = 6$ *and* $K_R = K_W \cdot K_M = 30$*. The codebook* $\mathcal{C}$ *is defined to be the entire set* $\mathfrak{S}_{3,2}$*. A* $(q = 3, z = 2, K_R = 30, r = 1)$ *RM rewrite coding scheme* $\{E_R, D_R\}$ *is constructed as follows:*

The ***encoding*** *algorithm* E_R *receives a message* $\boldsymbol{m} = (m_1, m_2) \in [K_W] \times [K_M]$ *and a state permutation* $\sigma \in \mathfrak{S}_{3,2}$*, and returns a permutation* π *in* $B_{3,2,1}(\sigma) \cap D_R^{-1}(\boldsymbol{m})$ *to store in the memory. It is constructed as follows:*

1: $\pi(1) \Leftarrow E_W(m_1, U_{1,2}(\sigma))$

2: $\pi_{[2:3]} \Leftarrow h_M(m_2)$

The ***decoding*** *algorithm* D_R *receives the stored permutation* $\pi \in \mathfrak{S}_{3,2}$*, and returns the stored message* $\boldsymbol{m} = (m_1, m_2) \in [K_W] \times [K_M]$*. It is constructed as follows:*

1: $m_1 \Leftarrow D_W(\pi(1))$

2: $m_2 \Leftarrow h_M^{-1}(\pi_{[2:3]})$

The rate of the code D_R is

$$R_R = (1/n)\log_2(K_R) = (1/6)\log(30) \approx 0.81.$$

The rate can be increased up to 2 bits per cell while keeping $r = 1$, by increasing z and q. We continue by increasing the parameter z.

3.1.2 Generalizing the Parameter z

In this subsection we generalize the construction to arbitrary values for the number of cells in each rank, z. The code for the second message part, h_M, generalizes naturally for any value of z, by taking M to be the multiset $M = \{2^z, 3^z\}$. Since z now can be large, it is important to choose the bijective functions h_M and h_M^{-1} such that they could be computed

efficiently. Luckily, several such efficient schemes exist in the literature, such as the scheme described in [60].

The code D_W for the part m_1, on the contrary, does not generalize naturally, since D_W in Example 3.1 does not have a natural generalization. To obtain such a generalization, we think of the characteristic vectors of the subsets of interest. The characteristic vector of $U_{1,2}(\sigma)$ is denoted as $\boldsymbol{s} = \theta_n(U_{1,2}(\sigma))$ (where $n = qz$), and is referred to as the state vector. The vector $\boldsymbol{x} = \theta_n(\pi(1))$ is called the codeword. The constraint $\pi(1) \subset U_{1,2}(\sigma)$ is then translated to the constraint $\boldsymbol{x} \leq \boldsymbol{s}$, which means that for each $j \in [n]$ we must have $x_j \leq s_j$. We now observe that this coding problem is similar to a concept known in the literature as Write-Once Memory codes, or WOM codes (see, for example, [69, 86]). In fact, the codes needed here are WOM codes for which the Hamming weight (number of non-zero bits) of the codewords is constant. Therefore, we say that D_W needs to be a "constant-weight WOM code". We use the word 'weight' from now on to denote the Hamming weight of a vector.

We define next the requirements of D_W in a vector notation. For a positive integer n and a real number $w \in [0, 1]$, we let $J_w(n) \subset \{0, 1\}^n$ be the set of all vectors of n bits whose weight equals $\lfloor wn \rfloor$. We use the name "constant-weight *strong* WOM code", since we will need to use a weaker version of this definition later. The weight of $\boldsymbol{s}$ in D_W is $2n/3$, and the weight of $\boldsymbol{x}$ is $n/3$. However, we allow for more general weight in the following definition, in preperation for the generalization of the number of ranks, q.

Definition 3.1: ***(Constant-weight strong WOM codes)*** *Let K_W and n be positive integers and let w_s be a real number in $[0, 1]$ and w_x be a real number in $[0, w_s]$. A surjective function $D_W : J_{w_x}(n) \to [K_W]$ is an (n, K_W, w_s, w_x)* ***constant-weight strong WOM code*** *if for each message $m \in [K_W]$ and state vector $\boldsymbol{s} \in J_{w_s}(n)$, there exists a codeword vector $\boldsymbol{x} \leq \boldsymbol{s}$ in the subset $D_W^{-1}(m) \subseteq J_{w_x}(n)$. The rate of a constant-weight strong WOM code is defined as $R_W = (1/n) \log K_W$.*

The code D_W in Example 3.1 is a $(n = 6, K_W = 5, w_s = 2/3, w_x = 1/3)$ constant-weight strong WOM code. It is useful to know the tightest upper bound on the rate of constant-weight strong WOM codes, which we call the capacity of those codes.

Proposition 3.1: *Let w_s and w_x be as defined in Definition* 3.1. *Then the capacity of constant-weight strong WOM codes is*

$$C_W = w_s H(w_x/w_s).$$

The proof of Proposition 3.1 is brought in Section 3.5. We also define the notions of coding scheme, capacity achieving, and efficient family for constant-weight strong WOM codes in the same way we defined it for RM rewriting codes. To construct capacity-achieving RM rewriting codes, we will need to use capacity-acheving constant-weight WOM codes as ingredients codes. However, we do not know how to construct an efficient capacity-achieving family of constant-weight strong WOM coding schemes. Therefore, we will present later a weaker notion of WOM codes, and show how to use it for the construction of RM rewriting codes.

3.1.3 Generalizing the Number of Ranks q

We continue with the generalization of the construction, where the next parameter to generalize is the number of ranks q. So the next scheme has general parameters q and z, while the cost constraint r is still kept at $r = 1$. In this case, we divide the message into $q - 1$ parts, m_1 to m_{q-1}. The encoding now starts in the same way as in the previous case, with the encoding of the part m_1 into the set $\pi(1)$, using a constant-weight strong WOM code. However, the parameters of the WOM code need to be slightly generalized. The numbers of cells now is $n = qz$, and E_W still chooses z cells for rank 1 of π out of the $2z$ cells of ranks 1 and 2 of σ. So we need a WOM code with the parameters $w_s = 2/q$ and $w_x = 1/q$.

The next step is to encode the message part m_2 into rank 2 of π. We can perform this encoding using the same WOM code D_W that was used for m_1. However, there is a difference now in the identity of the cells that are considered for occupying the set $\pi(2)$. In m_1, the cells that were considered as candidates to occupy $\pi(1)$ were the $2z$ cells in the set $U_{1,2}(\sigma)$, since all of these cell could be placed in $\pi(1)$ without dropping their rank (from σ to π) by

more then 1. In the encoding of m_2, we choose cells for rank 2 of π, so the z cells from rank 3 of σ can also be considered. Another issue here is that the cells that were already chosen for rank 1 of π should *not* be considered as candidates for rank 2. Taking these considerations into account, we see that the candidate cells for $\pi(2)$ are the z cells that were considered but not chosen for $\pi(1)$, together with the z cells in rank 3 of σ. Since these are two disjoint sets, the number of candidate cells for $\pi(2)$ is $2z$, the same as the number of candidates that we had for $\pi(1)$. The set of cells that were considered but not chosen for $\pi(1)$ are denoted by the set-theoretic difference $U_{1,2}(\sigma) \setminus \pi(1)$. Taking the union of $U_{1,2}(\sigma) \setminus \pi(1)$ with the set $\sigma(3)$, we get that the set of candidate cells to occupy rank 2 of π can be denoted by $U_{1,3}(\sigma) \setminus \pi(1)$.

Remark: In the coding of m_2, we can in fact use a WOM code with a shorter block length, since the cells in $\pi(1)$ do not need to take any part in the WOM code. This improves slightly the rate and computation complexity of the coding scheme. However, this improvement does not affect the asymptotic analysis we make in this chapter. Therefore, for the ease of presentation, we did not use this improvement.

We now apply the same idea to the rest of the sets of π, iteratively. On each iteration i from 1 to $q-2$, the set $\pi(i)$ must be a subset of $U_{1,i+1}(\sigma)$, to keep the cost at no more than 1. The sets $\{\pi(1), \ldots, \pi(i-1)\}$ were already determined in previous iterations, and thus their members cannot belong to $\pi(i)$. The set $U_{1,i-1}(\pi)$ contains the members of those sets (where $U_{1,0}(\pi)$ is the empty set). So we can say that the set $\pi(i)$ must be a subset of $U_{1,i+1}(\sigma) \setminus U_{1,i-1}(\pi)$. We let the state vector of the WOM code to be $\boldsymbol{s}_i = \theta_n(U_{1,i+1}(\sigma) \setminus U_{1,i-1}(\pi))$, and then use the WOM encoder $E_W(m_i, \boldsymbol{s}_i)$ to find an appropriate vector $\boldsymbol{x}_i \leq \boldsymbol{s}_i$ that represents m_i. We then assign $\pi(i) = \theta_n^{-1}(\boldsymbol{x}_i)$, such that $\pi(i)$ represents m_i.

If we use a capacity achieving family of constant-weight strong WOM codes, we store close to $w_s H(w_x/w_s) = 2(1/q)H(\frac{1}{2}) = 2/q$ bits per cell on each rank. Therefore, each of the $q-2$ message parts $m_1, \ldots, m_{q-2}$ can store close to $2/q$ bits per cell. So the RM rewriting code can store a total of $2(q-2)/q$ bits per cell, approaching the upper bound of 2 bits per cell (Corollary 2.2) when q grows. The last message part, m_{q-1}, is encoded with the same

code h_M that we used in the previous subsection for $q = 3$. The amount of information stored in the message m_{q-1} does not affect the asymptotic rate analysis, but is still beneficial.

To decode a message vector $\boldsymbol{m} = (m_1, m_2, \ldots, m_{q-1})$ from the stored permutation π, we can just decode each of the $q-1$ message parts separately. For each rank $i \in [q-2]$, the decoder finds the vector $\boldsymbol{x}_i = \theta_n(\pi(i))$, and then the message part m_i is calculated by the WOM decoder, $m_i \Leftarrow D_W(\boldsymbol{x}_i)$. The message part m_{q-1} is found by the decoder of the enumerative code, $m_{q-1} = h_M^{-1}(\pi_{[q-1:q]})$.

3.1.4 Generalizing the Cost Constraint r

We note first that if r is larger than $q-2$, the coding problem is trivial. When the cost constraint r is between 1 and $q-2$, the top $r+1$ cells of π can be occupied by *any* cell, since the magnitude of a drop from a rank at most q to a rank at least $q-r-1$, is at most r ranks. Therefore, we let the top $r+1$ ranks of π represents a single message part, named m_{q-r-1}. The message part m_{q-r-1} is mapped into the arraignment of the sequence of sets $(\pi(q-r), \pi(q-r+1), \ldots, \pi(q))$ by a generalization of the bijection h_M, defined by generalizing the multiset M into $M = \{(q-r)^z, (q-r+1)^z, \ldots, q^z\}$. The efficient coding scheme described in [60] for h_M and h_M^{-1} is suitable for any multiset M.

The rest of the message is divided into $q-r-1$ parts, m_1 to m_{q-r-1}, and their codes also need to generalized. The generalization of these coding scheme is also quite natural. First, consider the code for the message part m_1. When the cost constraint r is larger than 1, more cells are allowed to take rank 1 in π. Specifically, a cell whose rank in σ is at most $r+1$ and its rank in π is 1, drops by at most r ranks. Such drop does not cause the rewriting cost to exceed r. So the set of candidate cells for $\pi(1)$ in this case can be taken to be $U_{1,r+1}$. In the same way, for each i in $[1 : q-r-1]$, the set of candidate cells for $\pi(i)$ is $U_{1,i+r}(\sigma) \setminus U_{1,i-1}(\pi)$. The parameter w_s of the ingredient WOM is correspondingly generalized to $w_s = (r+1)/q$. This generalized algorithm is shown in Figure 3.1. We present now a formal description of the construction.

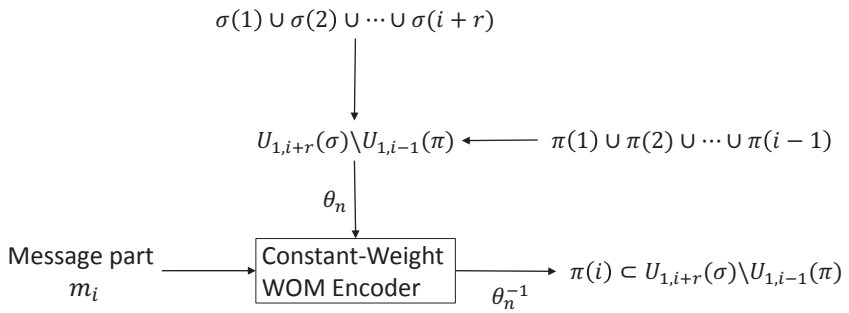

Figure 3.1: Iteration i of the encoding algorithm, where $1 \leq i \leq q-r-1$.

Construction 3.2: ***(A RM rewriting code from a constant-weight strong WOM code)*** *Let K_W, q, r, z be positive integers, let $n = qz$ and let (E_W, D_W) be an $(n, K_W, (r+1)/q, 1/q)$ constant-weight strong WOM coding scheme. Define the multiset $M = \{(q-r)^z, (q-r+1)^z, \ldots, q^z\}$ and let $K_M = |\mathfrak{S}_M|$ and $K_R = K_W^{q-r-1} \cdot K_M$. The codebook $\mathcal{C}$ is defined to be the entire set $\mathfrak{S}_{q,z}$. A (q, z, K_R, r) RM rewrite coding scheme $\{E_R, D_R\}$ is constructed as follows:*

The ***encoding*** *algorithm E_R receives a message $\boldsymbol{m} = (m_1, m_2, \ldots, m_{q-r}) \in [K_W]^{q-r-1} \times [K_M]$ and a state permutation $\sigma \in \mathfrak{S}_{q,z}$, and returns a permutation π in $B_{q,z,r}(\sigma) \cap D_R^{-1}(\boldsymbol{m})$ to store in the memory. It is constructed as follows:*

1: ***for*** $i = 1$ *to* $q-r-1$ ***do***
2: $\quad \boldsymbol{s}_i \Leftarrow \theta_n(U_{1,i+r}(\sigma) \setminus U_{1,i-1}(\pi))$
3: $\quad \boldsymbol{x}_i \Leftarrow E_W(m_i, \boldsymbol{s}_i)$
4: $\quad \pi(i) \Leftarrow \theta_n^{-1}(\boldsymbol{x}_i)$
5: ***end for***
6: $\pi_{[q-r:q]} \Leftarrow h_M(m_{q-r})$

The ***decoding*** *algorithm D_R receives the stored permutation $\pi \in \mathfrak{S}_{q,z}$, and returns the stored message $\boldsymbol{m} = (m_1, m_2, \ldots, m_{q-r}) \in [K_W]^{q-r-1} \times [K_M]$. It is constructed as follows:*

1: ***for*** $i = 1$ *to* $q-r-1$ ***do***
2: $\quad \boldsymbol{x}_i \Leftarrow \theta_n(\pi(i))$
3: $\quad m_i \Leftarrow D_W(\boldsymbol{x}_i)$

4: ***end for***

5: $m_{q-r} \Leftarrow h_M^{-1}(\pi_{[q-r:q]})$

Theorem 3.1: *Let* $\{E_W, D_W\}$ *be a member of an efficient capacity-achieving family of constant-weight strong WOM coding schemes. Then the family of RM rewrite coding schemes in Construction* 3.2 *is efficient and capacity-achieving.*

PROOF: The decoded message is equal to the encoded message by the property of the WOM code in Definition 3.1. By the explanation above the construction, it is clear that the cost is bounded by r, and therefore $\{E_R, D_R\}$ is a RM rewrite coding scheme. We will first show that $\{E_R, D_R\}$ is capacity achieving, and then show that it is efficient. Let $R_R = (1/n)\log K_R$ be the rate of a RM rewriting code. To show that $\{E_R, D_R\}$ is capacity achieving, we need to show that for any $\epsilon_R > 0$, $R_R > C_R - \epsilon_R$, for some q and z.

Since $\{E_W, D_W\}$ is capacity achieving, $R_W > C_W - \epsilon_W$ for any $\epsilon_W > 0$ and large enough n. Remember that $C_W = w_s H(w_x/w_s)$. In $\{E_R, D_R\}$ we use $w_s = (r+1)/q$ and $w_x = 1/q$, and so $C_W = \frac{r+1}{q} H\left(\frac{1}{r+1}\right)$. We have

$$\begin{aligned}
R_R &= (1/n)\log K_R \\
&= (1/n)\log(K_M \cdot K_W^{q-r-1}) \\
&> (q-r-1)(1/n)\log K_W \\
&> (q-r-1)(C_W - \epsilon_W) \\
&= (q-r-1)\left(\frac{r+1}{q} H\left(\frac{1}{r+1}\right) - \epsilon_W\right) \\
&= \frac{q-r-1}{q}(C_R - q\epsilon_W) \\
&= (C_R - q\epsilon_W)(1-(r+1)/q) \\
&> C_R - (r+1)^2/q - q\epsilon_W
\end{aligned} \tag{3.1}$$

The idea is to take $q = \lfloor (r+1)/\sqrt{\epsilon_W} \rfloor$ and $\epsilon_R = 3(r+1)\sqrt{\epsilon_W}$, and get that

$$R_R > C_R - \frac{(r+1)^2}{\lfloor (r+1)/\sqrt{\epsilon_W} \rfloor} - \lfloor (r+1)/\sqrt{\epsilon_W} \rfloor \epsilon_W > C_R - 2(r+1)\sqrt{\epsilon_W} - (r+1)\sqrt{\epsilon_W} = C_R - \epsilon_R.$$

Formally, we say: for any $\epsilon_R > 0$ and integer r, we set $\epsilon_W = \frac{\epsilon_R^2}{9(r+1)^2}$ and $q = \lfloor (r+1)/\sqrt{\epsilon_W} \rfloor$. Now if z is large enough then $n = qz$ is also large enough so that $R_W > C_W - \epsilon_W$, and then Equation 3.1 holds and we have $R_R > C_R - \epsilon_R$, proving that the construction is capacity achieving. Note that the family of coding schemes has a constant value of q and a growing value of z, as permitted by Definition 2.2 of capacity-achieving code families.

Next we show that $\{E_R, D_R\}$ is efficient. If the scheme (h_M, h_M^{-1}) is implemented as described in [60], then the time complexity of h_M and h_M^{-1} is polynomial in n. In addition, we assumed that E_W and D_W run in polynimial time in n. So since h_M and h_M^{-1} are executed only once in E_R and D_R, and E_W and D_W are executed less than q times in E_R and D_R, where $q < n$, we get that the time complexity of E_R and D_R is polynomial in n. ∎

3.1.5 How to Use Weak WOM Schemes

As mentioned earlier, we are not familiar with a family of efficient capacity-achieving constant-weight strong WOM coding schemes. Nonetheless, it turns out that we can construct efficient capacity-achieving WOM coding schemes that meet a slightly weaker definition, and use them to construct capacity-achieving RM rewriting codes. In this subsection we will define a weak notion of constant-weight WOM codes, and describe an associated RM rewriting coding scheme. In Sections 3.2 and 3.3 we will present yet weaker definition of WOM codes, together with constructions of appropriate WOM schemes and associated RM rewriting schemes.

In the weak definition of WOM codes, each codeword is a *pair*, composed of a constant-weight binary vector $\boldsymbol{x}$ and an index integer m_a. Meanwhile, the state is still a single vector $\boldsymbol{s}$, and the vector $\boldsymbol{x}$ in the codeword is required to be smaller than the state vector. We say that these codes are weaker since there is no restriction on the index integer in the

codeword. This allows the encoder to communicate some information to the decoder without restrictions.

Definition 3.2: ***(Constant-weight weak WOM codes)*** *Let K_W, K_a, n be positive integers and let w_s be a real number in $[0,1]$ and w_x be a real number in $[0, w_s]$. A surjective function $D_W : J_{w_x}(n) \times [K_a] \to [K_W]$ is an (n, K_W, K_a, w_s, w_x)* ***constant-weight weak WOM code*** *if for each message $m \in [K_W]$ and state vector $\boldsymbol{s} \in J_{w_s}(n)$, there exists a pair $(\boldsymbol{x}, m_a)$ in the subset $D_W^{-1}(m) \subseteq J_{w_x}(n) \times [K_a]$ such that $\boldsymbol{x} \leq \boldsymbol{s}$. The rate of a constant-weight weak WOM code is defined as $R_W = (1/n)\log(K_W/K_a)$.*

If $K_a = 1$, the code is in fact a constant-weight strong WOM code. We will only be interested in the case in which $K_W \gg K_a$. Since R_W is a decreasing function of K_a, it follows that the capacity of constant-weight weak WOM code is also $C_W = w_s H(w_x/w_s)$. Consider now the encoder E_R of a (q, z, K_R, r) RM rewriting code D_R with a codebook $\mathcal{C}$. For a message $m \in [K_R]$ and a state permutation $\sigma \in \mathcal{C}$, the encoder needs to find a permutation π in the intersection $B_{q,z,r}(\sigma) \cap D_R^{-1}(m)$. As before, we let the encoder determine the sets $\pi(1), \pi(2), \ldots, \pi(q-r-1)$ sequentially, such that each set $\pi(i)$ represents a message part m_i. If we were to use the previous encoding algorithm (in Construction 3.2) with a weak WOM code, the WOM encoding would find a pair $(\boldsymbol{x}_i, m_{a,i})$, and we could store the vector $\boldsymbol{x}_i$ by the set $\pi(i)$. However, we would not have a way to store the index $m_{a,i}$ that is also required for the decoding. To solve this, we will *add* some cells that will serve the sole purpose of storing the index $m_{a,i}$.

Since we use the WOM code $q-r-1$ times, once for each rank $i \in [q-r-1]$, it follows that we need to add $q-r-1$ different sets of cells. The added sets will take part in a larger permutation, such that the code will still meet Definition 2.1 of RM rewriting codes. To achieve that property, we let each added set of cells represent a permutation. That way the number of cells in each rank is constant, and a concatenation (in the sense of sting concatenation) of those permutations together results in a larger permutation. To keep the cost of rewriting bounded by r, we let each added set represent a permutation with $r+1$ ranks. That way each added set could be rewritten arbitrarily with a cost no greater than

r. We also let the number of cells in each rank in those added sets to be equal, in order to maximize the amount of stored information. Denote the number of cells in each rank in each of the added sets as a. Since each added set needs to store an index from the set $[K_a]$ with $r+1$ ranks, it follows that a must satisfy the inequality $|\mathfrak{S}_{r+1,a}| \geq K_a$. So to be economical with our resources, we set a to be the smallest integer that satisfies this inequality. We denote each of these additional permutations as $\pi_{a,i} \in \mathfrak{S}_{r+1,a}$. The main permutation is denoted by π_W, and the number of cells in each rank in π_W is denoted by z_W. The permutation π will be a string concatenation of the main permutation together with the $q-r-1$ added permutations. Note that this way the number of cells in each rank is not equal (there are more cells in the lowest $r+1$ ranks). This is actually not a problem, but it will be cleaner to present the construction if we add yet another permutation that "balances" the code. Specifically, we let π_b be a permutation of the multiset $\left\{(r+2)^{(q-r-1)a}, (r+3)^{(q-r-1)a}, \ldots, q^{(q-r-1)a}\right\}$ and let π^{-1} be the string concatenation $(\pi_{a,1}^{-1}, \ldots, \pi_{a,q-r-1}^{-1}, \pi_b^{-1}, \pi_W^{-1})$. This way in each rank there are exactly $z_W + (q-r-1)a$ cells. We denote $z = z_W + (q-r-1)a$, and then we get that π is a member of $\mathfrak{S}_{q,z}$.

On each iteration i from 1 to $q-r-1$ we use a constant-weight weak WOM code. The vectors $\boldsymbol{s}_i$ and $\boldsymbol{x}_i$ of the WOM code are now corresponding only to the main part of the permutation, and we denote their length by $n_W = qz_W$. We assign the state vector to be $\boldsymbol{s}_i = \theta_{n_W}(U_{1,i+r}(\sigma_W) \setminus U_{1,i-1}(\pi_W))$, where σ_W and π_W are the main parts of σ and π, accordingly. Note that $U_{1,i+r}(\sigma_W)$ and $U_{1,i-1}(\pi_W)$ are subsets of $[n_W]$ and that the characteristic vector θ_{n_W} is taken according to n_W as well. The message part m_i and the state vector $\boldsymbol{s}_i$ are used by the encoder E_W of an $(n_W, K_W, K_b, (r+1)/q, 1/q)$ constant-weight weak WOM code D_W. The result of the encoding is the pair $(\boldsymbol{x}_i, m_{a,i}) = E_W(m_i, \boldsymbol{s}_i)$. The vector $\boldsymbol{x}_i$ is stored on the main part of π by assigning $\pi_W(i) = \theta_{n_W}^{-1}(\boldsymbol{x}_i)$. The additional index $m_{a,i}$ is stored on the additional cells corresponding to rank i. Using an enumerative code $h_{r+1,a} : [|\mathfrak{S}_{r+1,a}|] \to \mathfrak{S}_{r+1,a}$, we assign $\pi_{a,i} = h_{r+1,a}(m_{a,i})$. After the lowest $q-r-1$ ranks of π_W are determined, we determine the highest $r+1$ ranks by setting $\pi_{W,[q-r,q]} = h_M(m_{q-r})$ where $M = \{(q-r)^{z_W}, (q-r+1)^{z_W}, \ldots, q^{z_W}\}$. Finally, the permutation π_b can be set

arbitrarily, say, to σ_b.

The decoding is performed in accordance with the encoding. For each rank $i \in [q-r-1]$, we first find $\boldsymbol{x}_i = \theta_{n_W}(\pi_W(i))$ and $m_{a,i} = h_{r+1,a}^{-1}(\pi_{a,i})$, and then assign $m_i = D_W(\boldsymbol{x}_i, m_{a,i})$. Finally, we assign $m_{q-r} = h_M^{-1}(\pi_{W,[q-r:q]})$.

Construction 3.3: ***(A RM rewriting code from a constant-weight weak WOM code)*** *Let K_W, K_a, q, r, and z_W be positive integers, and let $n_W = qz_W$. Let D_W be an $(n_W, K_W, K_a, (r+1)/q, 1/q)$ constant-weight weak WOM code with encoding algorithm E_W, and let a be the smallest integer for which $|\mathfrak{S}_{r+1,a}| \geq K_a$. Define the multiset $M = \{(q-r)^{z_W}, (q-r+1)^{z_W}, \ldots, q^{z_W}\}$ and let $K_M = |\mathfrak{S}_M|$ and $K = K_M \cdot K_W^{q-r-1}$.*

Let $z = z_W + (q-r-1)a$ and $n = qz$. Define a codebook $\mathcal{C} \subset \mathfrak{S}_{q,z}$ as a set of permutations $\pi \in \mathcal{C}$ in which π^{-1} is a string concatenation $(\pi_W^{-1}, \pi_{a,1}^{-1}, \ldots, \pi_{a,q-r-1}^{-1}, \pi_b^{-1})$ such that the following conditions hold:

1. *$\pi_W \in \mathfrak{S}_{q,z_W}$.*
2. *For each rank $i \in [q-r-1]$, $\pi_{a,i} \in \mathfrak{S}_{r+1,a}$.*
3. *π_b is a permutation of the multiset $\left\{(r+2)^{(q-r-1)a}, (r+3)^{(q-r-1)a}, \ldots, q^{(q-r-1)a}\right\}$.*

A (q, z, K_R, r) RM rewrite coding scheme $\{E_R, D_R\}$ is constructed as follows:

*The **encoding** function E_R receives a message $\boldsymbol{m} = (m_1, m_2, \ldots, m_{q-r}) \in [K_W]^{q-r-1} \times [K_M]$ and a state permutation $\sigma \in \mathcal{C}$, and finds a permutation π in $B_{q,z,r}(\sigma) \cap D_R^{-1}(\boldsymbol{m})$ to store in the memory. It is constructed as follows:*

1: ***for*** $i = 1$ *to* $q-r-1$ ***do***
2: $\quad \boldsymbol{s}_i \Leftarrow \theta_{n_W}(U_{1,i+r}(\sigma_W) \setminus U_{1,i-1}(\pi_W))$
3: $\quad (\boldsymbol{x}_i, m_{a,i}) \Leftarrow E_W(m_i, \boldsymbol{s}_i)$
4: $\quad \pi_W(i) \Leftarrow \theta_{n_W}^{-1}(\boldsymbol{x}_i)$
5: $\quad \pi_{a,i} \Leftarrow h_{r+1,a}(m_{a,i})$
6: ***end for***
7: $\pi_{W,[q-r:q]} \Leftarrow h_M(m_{q-r})$

8: $\pi_b \Leftarrow \sigma_b$

The ***decoding*** *function* D_R *receives the stored permutation* $\pi \in \mathcal{C}$*, and finds the stored message* $\boldsymbol{m} = (m_1, m_2, \ldots, m_{q-r}) \in [K_W]^{q-r-1} \times [K_M]$*. It is constructed as follows:*

1: ***for*** $i = 1$ *to* $q - r - 1$ ***do***

2: $\quad \boldsymbol{x}_i \Leftarrow \theta_{n_W}(\pi_W(i))$

3: $\quad m_{a,i} \Leftarrow h^{-1}_{r+1,a}(\pi_{a,i})$

4: $\quad m_i \Leftarrow D_W(\boldsymbol{x}_i, m_{a,i})$

5: ***end for***

6: $m_{q-r} \Leftarrow h^{-1}_M(\pi_{W,[q-r:q]})$

Remark: To be more economical with our resources, we could use the added sets "on top of each other", such that the $r + 1$ lowest ranks store one added set, the next $r + 1$ ranks store another added set, and so on. To ease the presentation, we did not describe the construction this way, since the asymptotic performance is not affected. However, such a method could increase the performance of practical systems.

Theorem 3.2: *Let* $\{E_W, D_W\}$ *be a member of an efficient capacity-achieving family of constant-weight weak WOM coding schemes. Then the family of RM rewrite coding schemes in Construction* 3.3 *is efficient and capacity-achieving.*

The proof of Theorem 3.2 is similar to that of Theorem 3.1 and is brought in Section 3.5.

3.2 Constant-Weight Polar WOM Codes

In this section we consider the use of polar WOM schemes [7] for the construction of constant-weight weak WOM schemes. Polar WOM codes do not have a constant weight, and thus require a modification in order to be used in Construction 3.3 of RM rewriting codes. The modification we propose in this section is exploiting the fact that while polar WOM codes do not have a constant weight, their weight is still *concentrated* around a constant value. This section is composed of two subsections. In the first, we show a general method to

construct constant-weight weak WOM codes from WOM codes with concentrated weight. The second subsection describes the construction of polar WOM schemes of Burshtein and Strugatski [7], and explains how they could be used as concentrated-weight WOM schemes.

3.2.1 Constant-Weight Weak WOM Schemes from Concentrated-Weight Strong Schemes

We first introduce additional notation. Label the weight of a vector $\boldsymbol{x}$ by $w_H(\boldsymbol{x})$. For $\delta > 0$, let $J_{w_x}(n, \delta)$ be the set of all n-bit vectors $\boldsymbol{x}$ such that $|w_x - w_H(\boldsymbol{x})/n| \leq \delta$.

Definition 3.3: ***(Concentrated-weight WOM codes)*** *Let K_C and n be positive integers and let w_s be in $[0, 1]$, w_x be in $[0, w_s]$, and δ in $[0, 1]$. A surjective function $D_C : J_{w_x}(n, \delta) \to [K_C]$ is a $(n, K_C, w_s, w_x, \delta)$* ***concentrated-weight WOM code*** *if for each message $m \in [K_C]$ and state vector $\boldsymbol{s} \in J_{w_s}(n)$ there exists a vector $\boldsymbol{x} \leq \boldsymbol{s}$ in the subset $D_C^{-1}(m) \subseteq J_{w_x}(n, \delta)$.*

From Theorem 1 in [36] and Proposition 3.1 we get that the capacity of concentrated-weight WOM codes in $C_W = w_s H(w_x/w_s)$. We define the notion of efficient capacity-achieving family of concentrated-weight WOM coding schemes accordingly. For the construction of constant-weight weak WOM codes from concentrated-weight WOM codes, we will use another type of enumerative coding schemes. For an integer n and δ in $[0, 1/2]$, let $J_{\leq\delta}(n)$ be the set of all n-bit vectors of weight at most δn, and define some bijective function $h_{\leq\delta} : \left[\sum_{j=1}^{\lfloor \delta n \rfloor} \binom{n}{j}\right] \to J_{\leq\delta}(n)$ with an inverse function $h_{\leq\delta}^{-1}$. The enumeration scheme $(h_{\leq\delta}, h_{\leq\delta}^{-1})$ can be implemented with computational complexity polynomial in n by methods such as [4, pp. 27-30], [67, 83].

We will now describe a construction of a constant-weight weak WOM coding scheme from a concentrated-weight WOM coding scheme. We start with the encoder E_W of the constant-weight weak WOM codes. According to Definition 3.2, given a message $m \in [K_W]$ and a state $\boldsymbol{s} \in J_{w_s}(n)$, the encoder needs to find a pair $(\boldsymbol{x}, m_a)$ in the set $D_W^{-1}(m)$ such that $\boldsymbol{x} \leq \boldsymbol{s}$. We start the encoding by finding the vector $\boldsymbol{x}_C = E_C(m, \boldsymbol{s})$ by the encoder

of an $(n, K_C, w_s, w_x, \delta)$ concentrated-weight WOM code. We know that the weight of $\boldsymbol{x}_C$ is "δ-close" to $w_x n$, but we need to find a vector with weight exactly $\lfloor w_x n \rfloor$. To do this, the main idea is to "flip" $|\lfloor w_x n \rfloor - w_H(\boldsymbol{x}_C)|$ bits in $\boldsymbol{x}_C$ to get a vector $\boldsymbol{x} \leq \boldsymbol{s}$ of weight $\lfloor w_x n \rfloor$, and store the location of the flipped bits in m_a. Let $\boldsymbol{a}$ be the n-bit vector of the flipped locations, such that $\boldsymbol{x} = \boldsymbol{x}_C \oplus \boldsymbol{a}$, where $\oplus$ is the bitwise XOR operation. It is clear that the weight of $\boldsymbol{a}$ must be $|\lfloor w_x n \rfloor - w_H(\boldsymbol{x}_C)|$. Let $\boldsymbol{x}_C = (x_{C,1}, x_{C,2}, \ldots, x_{C,n})$. If $w_H(\boldsymbol{x}_C) < w_x n$, we also must have $a_i = 0$ wherever $x_{C,i} = 1$, since we only want to flip 0's to 1's to increase the weight. In addition, we must have $a_i = 0$ wherever $s_i = 0$, since in those locations we have $x_{C,i} = 0$ and we want to get $x_i \leq s_i$. We can summarize those conditions by requiring that $\boldsymbol{a} \leq \boldsymbol{s} \oplus \boldsymbol{x}_C$ if $w_H(\boldsymbol{x}_C) < w_x n$. In the case that $w_H(\boldsymbol{x}_C) > w_x n$, we should require that $\boldsymbol{a} \leq \boldsymbol{x}_C$, since a_i can be 1 only where $x_{C,i} = 1$. In both cases we have the desired properties $\boldsymbol{x} \leq \boldsymbol{s}$, $w_H(\boldsymbol{x}) = \lfloor w_x n \rfloor$ and $w_H(\boldsymbol{a}) \leq \delta n$.

To complete the encoding, we let m_a be the index of the vector $\boldsymbol{a}$ in an enumeration of the n-bit vectors of weight at most δn. That will minimize the space required to store $\boldsymbol{a}$. Using an enumerative coding scheme, we assign $m_a = h^{-1}_{\leq \delta}(\boldsymbol{a})$. The decoding is now straightforward, and is described in the following formal description of the construction.

Construction 3.4: ***(A constant-weight weak WOM code from a concentrated-weight WOM code)*** *Let K_C and n be positive integers and let w_s be in $[0, 1]$, w_x be in $[0, w_s]$ and δ in $[0, 1/2]$. Let D_C be an $(n, K_C, w_s, w_x, \delta)$ concentrated-weight WOM code, and define $K_W = K_C$ and $K_a = \sum_{i=0}^{\lfloor \delta n \rfloor} \binom{n}{i}$.*

An (n, K_W, K_a, w_s, w_x) constant-weight weak WOM coding scheme $\{E_W, D_W\}$ is defined as follows:

The ***encoding*** *function E_W receives a message $m \in [K_W]$ and a state vector $\boldsymbol{s} \in J_{w_x}(n)$, and finds a pair $(\boldsymbol{x}, m_a)$ in $D_W^{-1}(m) \subseteq J_{w_x}(n) \times [K_a]$ such that $\boldsymbol{x} \leq \boldsymbol{s}$. It is constructed as follows:*

1. *Let $\boldsymbol{x}_C \Leftarrow E_C(\boldsymbol{s}, m)$.*

2. *Let $\boldsymbol{a}$ be an arbitrary vector of weight $|\lfloor w_x n \rfloor - w_H(\boldsymbol{x}_C)|$ such that $\boldsymbol{a} \leq \boldsymbol{s} \oplus \boldsymbol{x}_C$ if*

$w_H(\boldsymbol{x}_C) \leq w_x n$ *and* $\boldsymbol{a} \leq \boldsymbol{x}_C$ *otherwise.*

3. *Return the pair* $(\boldsymbol{x}, m_a) \Leftarrow (\boldsymbol{x}_C \oplus \boldsymbol{a}, h_{\leq\delta}^{-1}(\boldsymbol{a}))$.

The ***decoding*** *function* D_W *receives the stored pair* $(\boldsymbol{x}, m_a) \in J_{w_x}(n) \times [K_a]$*, and finds the stored message* $m \in [K_W]$*. It is constructed as follows:*

1. *Let* $\boldsymbol{a} \Leftarrow h_{\leq\delta}(m_a)$.

2. *Let* $\boldsymbol{x}_C \Leftarrow \boldsymbol{x} \oplus \boldsymbol{a}$.

3. *Return* $m \Leftarrow D_C(\boldsymbol{x}_C)$.

Theorem 3.3: *Let* $\{E_C, D_C\}$ *be a member of an efficient capacity-achieving family of concentrated-weight WOM coding schemes. Then Construction* 3.4 *describes an efficient capacity-achieving family of constant-weight weak WOM coding schemes for a sufficiently small* δ.

PROOF: First, since $E_C, D_C, h_{\leq\delta}$, and $h_{\leq\delta}^{-1}$ can be performed in polynomial time in n, it follows directly that E_W and D_W can also be performed in polynomial time in n. Next, we show that the family of coding schemes is capacity achieving. For large enough n we have $(1/n)\log K_W > C_W - \epsilon_C$. So

$$R_W = (1/n)\log(K_W/K_a) > C_W - \epsilon_C - H(\delta),$$

since $\log K_a = \log \sum_{i=0}^{\lfloor \delta n \rfloor} \binom{n}{i} \leq nH(\delta)$ by Stirling's formula. Now, given $\epsilon_W > 0$, we let $\epsilon_C = \epsilon_W/2$ and $\delta = H^{-1}(\epsilon_W/2)$ such that $\epsilon_C + H(\delta) = \epsilon_W$. So for large enough n we have $R_W > C_W - \epsilon_C - H(\delta) = C_W - \epsilon_W$. ∎

3.2.2 Polar WOM Codes

There are two properties of polar WOM coding schemes that do not fit well in our model. First, the scheme requires the presence of common randomness, known both to the encoder

and to the decoder. Such an assumption brings some weakness to the construction, but can find some justification in practical applications such as flash memory devices. For example, the common randomness can be the address of the storage location within the device. Second, the proposed encoding algorithm for polar WOM coding schemes does not always succeed in finding a correct codeword for the encoded message. In particular, the algorithm is randomized, and it only guarantees to succeed with high probability over the algorithm randomness and the common randomness. Nonetheless, for flash memory application, this assumption can be justified by the fact that such failure probability is much smaller than the unreliable nature of the devices. Therefore, some error-correction capability must be included in the construction for such practical implementation, and a failure of the encoding algorithm will not significantly affect the decoding failure rate. More approaches to tackle this issue are described in [7].

The construction is based on the method of channel polarization, which was first proposed by Arikan in his seminal paper [1] in the context of channel coding. We describe it here briefly by its application for WOM coding. This application is based on the use of polar coding for lossy source coding that was proposed by Korada and Urbanke [50].

Let n be a power of 2, and let $G_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ and $G_2^{\otimes \log n}$ be its $\log n$-th Kronecker product. Consider a memoryless channel with a binary-input and transition probability $W(y|x)$. Define a vector $\boldsymbol{u} \in \{0,1\}^n$, and let $\boldsymbol{x} = \boldsymbol{u}G_2^{\otimes \log n}$, where the matrix multiplication is over $\mathbb{F}_2$. The vector $\boldsymbol{x}$ is the input to the channel, and $\boldsymbol{y}$ is the output vector. The main idea of polar coding is to define n sub-channels

$$W_n^{(i)}(\boldsymbol{y}, \boldsymbol{u}_{[i-1]}|u_i) = P(\boldsymbol{y}, \boldsymbol{u}_{[i-1]}|u_i) = \frac{1}{2^{n-1}} \sum_{\boldsymbol{u}_{[i+1:n]}} P(\boldsymbol{y}|\boldsymbol{u}).$$

For large n, each sub-channel is either very reliable or very noisy, and therefore it is said that the channel is polarized. A useful measure for the reliability of a sub-channel $W_n^{(i)}$ is

its Bhattacharyya parameter, defined by

$$Z(W_n^{(i)}) = \sum_{y \in \mathcal{Y}} \sqrt{W_n^{(i)}(y|0) W_n^{(i)}(y|1)}. \tag{3.2}$$

Consider now a write-once memory. Let $\boldsymbol{s} \in \{0,1\}^n$ be the state vector, and let w_s be the fraction of 1's in $\boldsymbol{s}$. In addition, assume that a user wishes to store the message $m \in K_C$ with a codeword $\boldsymbol{x} \in J_{w_x}(n, \delta)$. The following scheme allows a rate arbitrarily close to C_W for n sufficiently large. The construction uses a compression scheme based on a *test channel*. Let v be a binary input to the channel, and (s, g) be the output, where s and g are binary variables as well. Denote $x = g \oplus v$. The probability transition function of the channel is given by

$$W(s, g|v) = \begin{cases} w_s - w_x & \text{if } (s,x) = (1,0), \\ w_x & \text{if } (s,x) = (1,1), \\ 1 - w_s & \text{if } (s,x) = (0,0), \\ 0 & \text{if } (s,x) = (0,1). \end{cases}$$

The channel is polarized by the sub-channels $W_n^{(i)}$ of Equation 3.2, and a *frozen set* F is defined by

$$F = \left\{ i \in [n] : Z(W_n^{(i)}) \geq 1 - 2\delta_n^2 \right\},$$

where $\delta_n = 2^{-n^\beta}/(2n)$, for $0 < \beta < 1/2$. It is easy to show that the capacity of the test channel is $C_T = 1 - C_W$. It was shown in [50] that $|F| = n(C_T + \epsilon_C) = n(1 - C_W + \epsilon_C)$, where ϵ_C is arbitrarily small for n sufficiently large. Let $\boldsymbol{g}$ be a common randomness source from an n dimensional uniformly distributed random binary vector. The coding scheme is the following:

Construction 3.5: ***(A Polar WOM code [7])*** *Let n be a positive integer and let w_s be in $[0,1]$, w_x be in $[0, w_s]$ and δ in $[0, 1/2]$. Let ϵ_C be in $[0, 1/2]$ such that $K_C = 2^{n(C_W - \epsilon_C)}$ is an integer.*

The ***encoding*** *function E_C receives a message $\boldsymbol{m} \in \{0,1\}^{\lceil \log K_C \rceil}$, a state vector $\boldsymbol{s} \in$*

$J_{w_s}(n)$ and the dither vector $\boldsymbol{g} \in \{0,1\}^n$, and returns a vector $\boldsymbol{x} \leq \boldsymbol{s}$ in $D_C^{-1}(\boldsymbol{m}) \subseteq J_{w_x}(n,\delta)$ with high probability. It is constructed as follows:

1. *Assign $y_j = (s_j, g_j)$ and $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)$.*
2. *Define a vector $\boldsymbol{u} \in \{0,1\}^n$ such that $\boldsymbol{u}_F = \boldsymbol{m}$.*
3. *Create a vector $\hat{\boldsymbol{u}} \in \{0,1\}^n$ by compressing the vector $\boldsymbol{y}$ according to the following successive cancellation scheme: For $i = 1, 2, \ldots, n$, let $\hat{u}_i = u_i$ if $i \in F$. Otherwise, let*

$$\hat{u}_i = \begin{cases} 0 & w.p.\ L_n^{(i)}/(L_n^{(i)}+1) \\ 1 & w.p.\ 1/(L_n^{(i)}+1) \end{cases},$$

where w.p. denotes with probability and

$$L_n^{(i)} = L_n^{(i)}(\boldsymbol{y}, \hat{\boldsymbol{u}}_{[i-1]}) = \frac{W_n^{(i)}(\boldsymbol{y}, \hat{\boldsymbol{u}}_{[i-1]} | u_i = 0)}{W_n^{(i)}(\boldsymbol{y}, \hat{\boldsymbol{u}}_{[i-1]} | u_i = 1)}.$$

4. *Assign $\boldsymbol{v} \Leftarrow \hat{\boldsymbol{u}} G_2^{\otimes \log n}$.*
5. *Return $\boldsymbol{x} \Leftarrow \boldsymbol{v} \oplus \boldsymbol{g}$.*

*The **decoding** function D_C receives the stored vector $\boldsymbol{x} \in J_{w_x}(n,\delta)$ and the dither vector $\boldsymbol{g} \in \{0,1\}^n$, and finds the stored message $\boldsymbol{m} \in \{0,1\}^{\lceil \log K_C \rceil}$. It is constructed as follows:*

1. *Assign $\boldsymbol{v} \Leftarrow \boldsymbol{x} \oplus \boldsymbol{g}$.*
2. *Assign $\hat{\boldsymbol{u}} \Leftarrow \boldsymbol{v}(G_2^{\otimes \log n})^{-1}$.*
3. *Return $\boldsymbol{m} \Leftarrow \hat{\boldsymbol{u}}_F$.*

In [7], a few slight modifications for this scheme are described, for the sake of the proof. We use the coding scheme (E_C, D_C) of Construction 3.5 as an $(N, K_C, w_s, w_x, \delta)$ concentrated-weight WOM coding scheme, even though it does not meet the definition precisely.

By the proof of Lemma 1 of [7], for $0 < \beta < 1/2$, the vector $\boldsymbol{x}$ found by the above encoding algorithm is in $D_C^{-1}(\boldsymbol{m})$ and in $J_{w_x}(n, \delta)$ w.p. at least $1 - 2^{-n^\beta}$ for n sufficiently large. Therefore, the polar WOM scheme of Construction 3.5 can be used as a practical concentrated-weight WOM coding scheme for the construction of RM rewriting codes by Constructions 3.3 and 3.4. Lemma 1 of [7] also proves that this scheme is capacity achieving. By the results in [50], the encoding and the decoding complexities are $O(n \log n)$, and therefore the scheme is efficient. This completes our first full description of a RM rewrite coding scheme in this chapter, although it does not meet the definitions of Section 2.2 precisely. In the next section we describe a construction of efficient capacity-achieving RM rewrite coding schemes that meet the definitions of Section 2.2.

3.3 Rank-Modulation Schemes from Hash WOM Schemes

The construction in this section is based on a recent construction of WOM codes by Shpilka [79]. This will require an additional modification to Construction 3.3 of RM rewrite coding schemes.

3.3.1 Rank-Modulation Schemes from Concatenated WOM Schemes

The construction of Shpilka does not meet any of our previous definitions of WOM codes. Therefore, we define yet another type of WOM codes, called "constant-weight concatenated WOM codes". As the name implies, the definition is a string concatenation of constant-weight WOM codes.

Definition 3.4: ***(Constant-weight concatenated WOM codes)*** *Let $K_W, K_a, n,$ and t be positive integers and let w_s be a real number in $[0, 1]$ and w_x be a real number in $[0, w_s]$. A surjective function $D_W : (J_{w_x}(n))^t \times [K_a] \to [K_W]$ is an $(n, t, K_W, K_a, w_s, w_x)$* ***constant-weight concatenated WOM code*** *if for each message $m \in [K_W]$ and state vector $\boldsymbol{s} \in (J_{w_s}(n))^t$, there exists a pair $(\boldsymbol{x}, m_a)$ in the subset $D_W^{-1}(m) \subseteq (J_{w_x}(n))^t \times [K_b]$ such that $\boldsymbol{x} \leq \boldsymbol{s}$.*

Note that the block length of constant-weight concatenated WOM codes is nt, and therefore their rate is defined to be $R_W = \frac{1}{nt}\log K_W$. Since concatenation does not change the code rate, the capacity of constant-weight concatenated WOM codes is $C_W = w_s H(w_x/w_s)$. We define the notion of coding schemes, capacity achieving and efficient family of schemes accordingly. Next, we use constant-weight concatenated WOM coding schemes to construct RM rewrite coding schemes by a similar concatenation.

Construction 3.6: ***(A RM rewriting scheme from a constant-weight concatenated WOM scheme)*** *Let K_W, K_a, q, r, t, and z_W be positive integers, and let $n_W = qz_W$. Let D_W be an $(n_W, t, K_W, K_a, (r+1)/q, 1/q)$ constant-weight concatenated WOM code with encoding algorithm E_W, and let a be the smallest integer for which $|\mathfrak{S}_{r+1,a}| \geq K_b$. Define the multiset $M = \{(q-r)^{z_W}, (q-r+1)^{z_W}, \ldots, q^{z_W}\}$ and let $K_M = |\mathfrak{S}_M|$ and $K_R = K_M \cdot K_W^{q-r-1}$.*

Let $z = tz_W + (q-r-1)a$ and $n = qz$. Define a codebook $\mathcal{C} \subset \mathfrak{S}_{q,z}$ as a set of permutations $\pi \in \mathcal{C}$ in which π^{-1} is a string concatenation $(\pi_{a,1}^{-1}, \ldots, \pi_{a,q-r-1}^{-1}, \pi_b^{-1}, \pi_{x,1}^{-1}, \ldots, \pi_{x,t}^{-1})$ such that the following conditions hold:

1. *$\pi_{x,i} \in \mathfrak{S}_{q,z_W}$ for each $i \in [t]$.*
2. *$\pi_{a,i} \in \mathfrak{S}_{r+1,a}$ for each rank $i \in [q-r-1]$.*
3. *π_b is a permutation of the multiset $\left\{(r+2)^{(q-r-1)a}, (r+3)^{(q-r-1)a}, \ldots, q^{(q-r-1)a}\right\}$.*

Denote the string concatenation $(\pi_{x,1}^{-1}, \ldots, \pi_{x,t}^{-1})$ by π_W^{-1}, and denote σ_W in the same way. A (q, z, K_R, r) RM rewrite coding scheme $\{E_R, D_R\}$ is constructed as follows:

The ***encoding*** *function E_R receives a message $\boldsymbol{m} = (m_1, m_2, \ldots, m_{q-r}) \in [K_W]^{q-r-1} \times [K_M]$ and a state permutation $\sigma \in \mathcal{C}$, and finds a permutation π in $B_{q,z,r}(\sigma) \cap D_R^{-1}(\boldsymbol{m})$ to store in the memory. It is constructed as follows:*

1: ***for*** $i = 1$ *to* $q-r-1$ ***do***
2: $\quad \boldsymbol{s}_i \Leftarrow \theta_{n_W}(U_{1,i+r}(\sigma_W) \setminus U_{1,i-1}(\pi_W))$
3: $\quad (\boldsymbol{x}_i, m_{a,i}) \Leftarrow E_W(m_i, \boldsymbol{s}_i)$

4: $\pi_W(i) \Leftarrow \theta_{n_W}^{-1}(\boldsymbol{x}_i)$

5: $\pi_{a,i} \Leftarrow h_{r+1,a}(m_{a,i})$

6: ***end for***

7: $\pi_{W,[q-r,q]} \Leftarrow h_M(m_{q-r})$

8: $\pi_b = \sigma_b$

The ***decoding*** *function D_R receives the stored permutation $\pi \in \mathcal{C}$, and finds the stored message $\boldsymbol{m} = (m_1, m_2, \ldots, m_{q-r}) \in [K_W]^{q-r-1} \times [K_M]$. It is constructed as follows:*

1: ***for*** $i = 1$ *to* $q - r - 1$ ***do***

2: $\boldsymbol{x}_i \Leftarrow \theta_{n_W}(\pi_W(i))$

3: $m_{a,i} \Leftarrow h_{r+1,a}^{-1}(\pi_{a,i})$

4: $m_i \Leftarrow D_W(\boldsymbol{x}_i, m_{a,i})$

5: ***end for***

6: $m_{q-r} \Leftarrow h_M^{-1}(\pi_{W,[q-r,q]})$

Since again concatenation does not affect the rate of the code, the argument of the proof of Theorem 3.2 gives the following statement:

Theorem 3.4: *Let $\{E_W, D_W\}$ be a member of an efficient capacity-achieving family of constant-weight concatenated WOM coding schemes. Then the family of RM rewrite coding schemes in Construction* 3.6 *is efficient and capacity-achieving.*

3.3.2 Hash WOM Codes

In [79] Shpilka proposed a construction of efficient capacity-achieving WOM coding scheme. The proposed scheme follows the concatenated structure of Definition 3.4, but does not have a constant weight. In this subsection we describe a slightly modified version of the construction of Shpilka, that does exhibit a constant weight.

To describe the construction, we follow the definitions of Shpilka [79]. The construction is based on a set of hash functions. For positive integers n, k, l and field members $a, b \in \mathbb{F}_{2^n}$, define a map $H_{a,b}^{n,k,l} : \{0,1\}^n \to \{0,1\}^{k-l}$ as $H_{a,b}^{n,k,l}(\boldsymbol{x}) = (ax + b)_{[k-l]}$. This notation means

that we compute the affine transformation $ax + b$ in $\mathbb{F}_{2^n}$, represent it as a vector of n bits using the natural map, and then keep the first $k - l$ bits of this vector. We represent this family of maps by $\mathcal{H}^{n,k,l}$, namely

$$\mathcal{H}^{n,k,l} = \left\{ H^{n,k,l}_{a,b} | a, b \in \mathbb{F}_{2^n} \right\}.$$

The family $\mathcal{H}^{n,k,l}$ contains 2^{2n} functions. For an integer $m_a \in [2^{2n}]$, we let H_{m_a} be the m_a-th function in $\mathcal{H}^{n,k,l}$.

Construction 3.7: ***(A constant-weight concatenated WOM coding scheme from hash functions)*** *Let ϵ, δ be in $[0, 1/2]$, w_s in $[0, 1]$, w_x in $[0, w_s]$ and $c > 20$. Let $n = \lceil (c/\epsilon) \log(1/\epsilon) \rceil$, $k = \lfloor n(C_W - 2\epsilon/3) \rfloor$, $t_1 = \lfloor (1/\epsilon)^{c/12} - 1 \rfloor$, and $t_2 = 2^{\frac{4n}{\delta}}$. Finally, Let $t = t_1 t_2$, $K_b = 2^k$, and $K_a = 2^{2n}$. A $(n, t, K_b^t, K_a^{t_2}, w_s, w_x)$ constant-weight concatenated WOM code is defined as follows:*

The ***encoding*** *function E_W receives a message matrix $\boldsymbol{m} \in [K_b]^{t_1 \times t_2}$, a state matrix of vectors $\boldsymbol{s} \in (J_{w_s}(n))^{t_1 \times t_2}$, and returns a pair $(\boldsymbol{x}, \boldsymbol{m}_a)$ in $D_W^{-1}(\boldsymbol{m}) \subseteq (J_{w_x}(n))^{t_1 \times t_2} \times [K_a]^{t_2}$ such that for each $(i, j) \in [t_1] \times [t_2]$ we have $\boldsymbol{x}_{i,j} \leq \boldsymbol{s}_{i,j}$. It is constructed as follows: for each $j \in [t_2]$, use a brute force search to find an index $m_{a,j} \in [K_a]$ and a vector $\boldsymbol{x}_j = (\boldsymbol{x}_{1,j}, \ldots, \boldsymbol{x}_{t_1,j})$ such that for all $i \in [t_1]$, the following conditions hold:*

1. $\boldsymbol{x}_{i,j} \leq \boldsymbol{s}_{i,j}$.
2. $\boldsymbol{x}_{i,j} \in J_{w_x}(n)$.
3. $H_{m_{a,j}}(\boldsymbol{x}_{i,j}) = m_{i,j}$.

The ***decoding*** *function D_W receives the stored pair $(\boldsymbol{x}, \boldsymbol{m}_a) \in (J_{w_x}(n))^{t_1 \times t_2} \times [K_a]$, and returns the stored message $\boldsymbol{m} \in [K_b]^{t_1 \times t_2}$. It is constructed as follows: for each pair $(i, j) \in [t_1] \times [t_2]$, assign $m_{i,j} \Leftarrow H_{m_{a,j}}(\boldsymbol{x}_{i,j})$.*

The only conceptual difference between Construction 3.7 and the construction in [79] is that here we require the vectors $\boldsymbol{x}_{i,j}$ to have a constant weight of $\lfloor w_x n \rfloor$, while the construction in [79] requires the weight of those vectors to be bounded only by $w_x n$. This difference is

crucial for the rank-modulation application, but in fact it has almost no effect on the proofs of the properties of the construction.

To prove that the code in Construction 3.7 is a constant-weight concatenated WOM code, we will need the following lemma from [79]:

Lemma 3.1: *[79, Corollary 2.3]: Let k', ℓ, t_1 and n be positive integers such that $\ell \leq k' \leq n$ and $t_1 < 2^{\ell/4}$. Let $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_{t_1} \subseteq \{0,1\}^n$ be sets of size $|\boldsymbol{X}_1|, \ldots, |\boldsymbol{X}_{t_1}| \geq 2^{k'}$. Then, for any $\boldsymbol{m}_1, \ldots, \boldsymbol{m}_{t_1} \in \{0,1\}^{k'-\ell}$ there exists $H_m \in \mathcal{H}^{n,k',\ell}$ and $\{\boldsymbol{x}_i \in \boldsymbol{X}_i\}$ such that for all $i \in [t_1]$, $H_m(\boldsymbol{x}_i) = \boldsymbol{m}_i$.*

Lemma 3.1 is proven using the leftover hash lemma [3, pp. 445], [5,39] and the probabilistic method.

Proposition 3.2: *The code D_W of Construction 3.7 is an $(n, t, K_b^t, K_a^{t_2}, w_s, w_x)$ constant-weight concatenated WOM code.*

Proof: The proof is almost the same as the proof of Lemma 2.4 in [79], except that here the codewords' weight is constant. Let $\ell = \lceil \epsilon n/3 \rceil$, $k' = k + \ell$ and

$$\boldsymbol{X}_i = \{\boldsymbol{x} \in \{0,1\}^n | \boldsymbol{x} \leq \boldsymbol{s}_i \text{ and } \boldsymbol{x} \in J_{w_x}(n)\}.$$

Since $\boldsymbol{x} \in J_{w_x}(n)$, we have that

$$|\boldsymbol{X}_i| = \binom{\lfloor w_s n \rfloor}{\lfloor w_x n \rfloor},$$

which by Stirling's formula can be lower bounded by

$$\begin{aligned} \geq 2^{w_s n H(w_x/w_s) - \log(w_s n)} &\geq 2^{nC_W - \log n} \\ &\geq 2^{nC_W - \epsilon n/3} = 2^{k'} \end{aligned}$$

For the last inequality we need $\epsilon n \geq 3 \log n$, which follows from

$$\frac{3 \log n}{\epsilon n} < \frac{3 \log[(2c/\epsilon) \log(1/\epsilon)]}{c \log(1/\epsilon)} < \frac{3 \log[(40/\epsilon) \log(1/\epsilon)]}{20 \log(1/\epsilon)} < 1.$$

Notice also that

$$t_1 = \lfloor (1/\epsilon)^{c/12} - 1 \rfloor < (1/\epsilon)^{c/12} = 2^{\frac{1}{4}\frac{\epsilon}{3}\frac{c}{\epsilon} \log(1/\epsilon)} \leq 2^{\frac{1}{4}\frac{\epsilon n}{3}} \leq 2^{\ell/4}.$$

So all of the conditions of Lemma 3.1 are met, which implies that the encoding of Construction 3.7 is always successful, and thus that D_W is a constant-weight concatenated WOM code. ∎

Theorem 3.5: *Construction* 3.7 *describes an efficient capacity-achieving family of concatenated WOM coding schemes.*

PROOF: We first show that the family is capacity achieving. We will need the following inequality:

$$\frac{2}{t_1} = \frac{2}{\lfloor (1/\epsilon)^{c/12} - 1 \rfloor} < 4\epsilon^{5/3} < \epsilon/3.$$

Now the rate can be bounded bellow as follows:

$$\begin{aligned} R_W &= \frac{t \log K_b - t_2 \log K_a}{nt} \\ &= \frac{t_1 \log K_b - \log K_a}{nt_1} \\ &= \frac{t_1 k - 2n}{nt_1} \\ &\geq \frac{t_1(C_W - 2\epsilon/3) - 2}{t_1} \\ &> C_W - 2\epsilon/3 - \epsilon/3 \\ &= C_W - \epsilon, \end{aligned}$$

and therefore the family is capacity achieving.

To show that the family is efficient, denote the block length of the code as $N = nt$. The encoding time is

$$t_2|\mathcal{H}^{n,k,\ell}| \cdot \sum_{i=1}^{t_1} |\boldsymbol{X}_i| \leq t_2 t_1 2^{3n} < t_2 2^{4n} = t_2^{1+\delta} < N^{1+\delta},$$

and the decoding time is

$$t_2 \cdot \text{poly}(kt_1 n) = 2^{4n/\delta}(2/\epsilon)^{O(c)} < N \cdot 2^{O(n\epsilon)} = N \cdot N^{O(\delta\epsilon)} = N^{1+O(\delta\epsilon)}.$$

This completes the proof of the theorem. ∎

Remark: Note that t_2 is exponential in $1/\epsilon$, and therefore the block length N is exponential in $(1/\epsilon)$. This can be an important disadvantage for these codes. In comparison, it is likely that the block length of polar WOM codes is only polynomial in $(1/\epsilon)$, since a similar result was shown recently in [30] for the case of polar lossy source codes, on which polar WOM codes are based.

We also note here that it is possible that the WOM codes of Gabizon and Shaltiel [25] could be modified for constant weight, to give RM rewriting codes with short block length without the dither and error probability of polar WOM codes.

3.4 Summary

In this part of the thesis we studied the limits of rank-modulation rewriting codes, and presented two capacity-achieving code constructions. The construction of Section 3.3, based on hash functions, has no possibility of error, but requires long blocklengths that might not be considered practical. On the other hand, the construction of section 3.2, based on polar codes, appears to have shorter blocklengths, but exhibits a small probability of error.

3.5 Capacity Proofs

Proof (of Proposition 3.1): The proof follows a similar proof by Heegard [36], for the case where the codewords' weight is not necessarily constant. Given a state $\boldsymbol{s}$, the number of vectors $\boldsymbol{x}$ of weight $\lfloor w_x n \rfloor$ such that $\boldsymbol{x} \leq \boldsymbol{s}$ is $\binom{\lfloor w_s n \rfloor}{\lfloor w_x n \rfloor}$. Since K_W cannot be greater than this number, we have

$$R_W = (1/n) \log K_W \leq (1/n) \log \binom{\lfloor w_s n \rfloor}{\lfloor w_x n \rfloor} \leq (1/n) \log 2^{w_s n H(w_x/w_s)} = C_W,$$

where the last inequality follows from Stirling's formula. Therefore, the capacity is at most C_W.

The lower bound on the capacity is proven by the probabilistic method. Randomly and uniformly partition $J_{w_x}(n)$ into K_W subsets of equal size,

$$|D_W^{-1}(m)| = |J_{w_x}(n)|/2^{nR_W}.$$

Fix $m \in [K_W]$ and $\boldsymbol{s} \in J_{w_s}(n)$, and let $\beta(\boldsymbol{s})$ be the set of vectors $\boldsymbol{x} \in J_{w_x}(n)$ such that $\boldsymbol{x} \leq \boldsymbol{s}$. Then

$$\begin{aligned} P(D_W^{-1}(m) \cap \beta(\boldsymbol{s}) = \emptyset) &= \prod_{i=0}^{|D_W^{-1}(m)|-1} \frac{|J_{w_x}(n)| - |\beta(\boldsymbol{s})| - i}{|J_{w_x}(n)| - i} \\ &\leq \left(\frac{|J_{w_x}(n)| - |\beta(\boldsymbol{s})|}{|J_{w_x}(n)|} \right)^{|D_W^{-1}(m)|}. \end{aligned}$$

$|\beta(\boldsymbol{s})| \geq 2^{nC_W - \log(w_s n)}$, and thus

$$\begin{aligned} P(D_W^{-1}(m) \cap \beta(\boldsymbol{s}) = \emptyset) &\leq (1 - |J_{w_x}(n)|^{-1} 2^{nC_W - \log(w_s n)})^{|J_{w_x}| 2^{-nR_W}} \\ &< e^{-(2^{n(C_W - R_W) - \log(w_s n)})}, \end{aligned}$$

where the last inequality follows from the fact that $(1-x)^y < e^{-xy}$ for $y > 0$. If $R_W < C_W$,

this probability vanishes for large n. In addition,

$$\begin{aligned}
&P(\exists m \in [K_W] \text{ and } \boldsymbol{s} \in J_{w_s}(n) \text{ s.t. } D_W^{-1}(m) \cap \beta(\boldsymbol{s}) = \emptyset) \\
&\qquad = P\left(\cup_{m\in[K_W]} \cup_{\boldsymbol{s}\in J_{w_s}(n)} \left\{D_W^{-1}(m) \cap \beta(\boldsymbol{s}) = \emptyset\right\}\right) \\
&\qquad \leq \sum_{m\in[K_W]} \sum_{\boldsymbol{s}\in J_{w_s}(n)} P(D_W^{-1}(m) \cap \beta(\boldsymbol{s}) = \emptyset) \\
&\qquad \leq 2^{n(R_W+H(w_s))} e^{-(2^{n(C_W-R_W)-\log(w_s n)})}.
\end{aligned}$$

This means that if $R_W < C_W$ and n is large enough, the probability that the partition is not a constant-weight strong WOM code approaches 0, and therefore there exists such a code, completing the proof. ∎

Proof (of Theorem 3.2): We will first show that $\{E_R, D_R\}$ is capacity achieving, and then show that it is efficient. Let $R_R = (1/n)\log K_R$ be the rate of a RM rewriting code. To show that $\{E_R, D_R\}$ is capacity achieving, we need to show that for any $\epsilon_R > 0$, $R_R > C_R - \epsilon_R$, for some q and z.

Since $\{E_W, D_W\}$ is capacity achieving, $R_W > C_W - \epsilon_W$ for any $\epsilon_W > 0$ and large enough n. Remember that $C_W = w_s H(w_x/w_s)$. In $\{E_R, D_R\}$ we use $w_s = (r+1)/q$ and $w_x = 1/q$, and so $C_W = \frac{r+1}{q} H\left(\frac{1}{r+1}\right)$. We will need to use the inequality $\log K_a > a$, which follows from:

$$\log K_a > \log|\mathfrak{S}_{r+1,a-1}| > \log|\mathfrak{S}_{2,a-1}| > 2a - 2 - \log 2a > a$$

Where the last inequality requires a to be at least 6. In addition, we will need the inequality $n_W/n > 1 - q^2\epsilon_W$, which follows from:

$$\begin{aligned}
\frac{n_W}{n} &= \frac{n_W}{n_W + q(q-r-1)a} > \frac{n_W}{n_W + q^2 a} > 1 - \frac{q^2 a}{n_W} > 1 - \frac{q^2 \log K_a}{n_W} \\
&= 1 - q^2\left(\frac{\log K_W}{n_W} - \frac{\log(K_W/K_a)}{n_W}\right) > 1 - q^2(C_W - (C_W - \epsilon_W)) = 1 - q^2\epsilon_W.
\end{aligned}$$

Now we can bound the rate from below, as follows:

$$\begin{aligned}
R_R &= (1/n)\log K_R \\
&= (1/n)\log(K_M \cdot K_W^{q-r-1}) \\
&> (q-r-1)(1/n)\log K_W \\
&> (q-r-1)(C_W - \epsilon_W)(n_W/n) \qquad (3.3)\\
&> (q-r-1)\left(\frac{r+1}{q}H\left(\frac{1}{r+1}\right) - \epsilon_W\right)(1-q^2\epsilon_W) \\
&= \frac{q-r-1}{q}(C_R - q\epsilon_W)(1-q^2\epsilon_W) \\
&= (C_R - q\epsilon_W)(1-(r+1)/q)(1-q^2\epsilon_W) \\
&> C_R - C_R q^2\epsilon_W - C_R(r+1)/q + (C_R(r+1)q\epsilon_W - q\epsilon_W) + (q^3\epsilon^2 - (r+1)q^2\epsilon_W^2) \\
&> C_R - (r+1)q^2\epsilon_W - (r+1)^2/q.
\end{aligned}$$

The idea is to take $q = \left\lfloor \left(\frac{r+1}{\epsilon_W}\right)^{1/3} \right\rfloor$ and $\epsilon_R = 3(r+1)^{2/3}\epsilon_W^{1/3}$ and get that

$$\begin{aligned}
R_R &> C_R - (r+1)\left\lfloor \left(\frac{r+1}{\epsilon_W}\right)^{1/3} \right\rfloor^2 \epsilon_W - \frac{(r+1)^2}{\left\lfloor \left(\frac{r+1}{\epsilon_W}\right)^{1/3} \right\rfloor} \\
&> C_R - (r+1)^{2/3}\epsilon_W^{1/3} - 2(r+1)^{2/3}\epsilon_W^{1/3} = C_R - \epsilon_R.
\end{aligned}$$

So we can say that for any $\epsilon_R > 0$ and integer r, we set $\epsilon_W = \frac{\epsilon_R^2}{9(r+1)^2}$ and $q = \lfloor (r+1)/\sqrt{\epsilon_W} \rfloor$. Now if z is large enough then $n = qz$ is also large enough so that $R_W > C_W - \epsilon_W$, and then Equation 3.3 holds and we have $R_R > C_R - \epsilon_R$.

Finally, we show that $\{E_R, D_R\}$ is efficient. If the scheme (h_M, h_M^{-1}) is implemented as described in [60], then the time complexity of h_M and h_M^{-1} is polynomial in n. In addition, we assumed that E_W and D_W run in polynomial time in n. So since h_M and h_M^{-1} are executed only once in E_R and D_R, and E_W and D_W are executed less than q times in E_R and D_R, where $q < n$, we get that the time complexity of E_R and D_R is polynomial in n. ∎

Part III

Write-Once Memory

Chapter 4

Rewriting with Sparse-Graph Codes

4.1 Rewriting and Erasure Quantization

4.1.1 Rewriting Model

In this chapter we consider a model that allows two writes on a block of n cells. A cell has a binary state chosen from $\{0, 1\}$, with the rewriting constraint that state 1 can be written to state 0, but not vice versa. All cells are initially set to be in state 1, and so there is no writing constraint for the first write. A vector is denoted by a bold symbol, such as $\boldsymbol{s} = (s_1, s_2, \ldots, s_n)$. The state of the n cells after the first write is denoted by the vector $\boldsymbol{s}$. We focus only on the second write, and we assume that after the first write, the state of the cells is i.i.d., where for each i, $\Pr\{s_i = 1\} = \beta$. We note that the special case of $\beta = 1/2$ is of practical importance, since it approximates the state after a normal page programming in flash memory[1]. The second write is concerned with how to store a message $\boldsymbol{m} \in \mathbb{F}_2^k$ by changing $\boldsymbol{s}$ to a new state $\boldsymbol{x}$ such that 1) the rewriting constraint is satisfied, and 2) $\boldsymbol{x}$ represents $\boldsymbol{m}$. This is achieved by the encoding operation of a rewriting code, defined formally in the following:

Definition 4.1: *A* rewriting code C_R *is a collection of disjoint subsets of* $\mathbb{F}_2^n$.

[1]In flash memory, the message to be written can be assumed to be random due to data compression and data randomization used in memory controllers.

Each element of C_R corresponds to a different message. Consider $M \in C_R$ that corresponds to a message $\boldsymbol{m}$, then for all $\boldsymbol{x} \in M$, we say that $\boldsymbol{x}$ is *labeled* by $\boldsymbol{m}$. The decoding function maps the set of labeled vectors into their labels, which are also the messages. To encode a message $\boldsymbol{m}$ given a state $\boldsymbol{s}$, the encoder needs to find a vector $\boldsymbol{x}$ with label $\boldsymbol{m}$ that can be written over $\boldsymbol{s}$. If the encoder does not find such vector $\boldsymbol{x}$, it declares a failure. The rewriting rate of C_R is defined by $R_{\text{WOM}} = k/n$. The rewriting capacity, which characterizes the maximum amount of information that can be stored per cell in the second write, is known to be β bits [36].

We are interested in rewriting codes with rates close to the capacity, together with efficient encoding algorithms with low failure probability. The main observation in the design of the proposed rewriting scheme of this chapter is that the rewriting problem is related to the problem of binary erasure quantization (BEQ), introduced in the next subsection.

4.1.2 Binary Erasure Quantization

The BEQ problem is concerned with the quantization of a binary *source sequence* $\boldsymbol{s}'$, for which some bits are erased. Formally, $\boldsymbol{s}' \in \{0, 1, *\}^n$, where $*$ represents erasures. $\boldsymbol{s}'$ needs to be quantized (compressed) such that every non-erased symbol of $\boldsymbol{s}'$ will maintain its value in the reconstructed vector. A reconstructed vector with such property is said to have *no distortion* from $\boldsymbol{s}'$. In this chapter we use linear BEQ codes, defined as follows:

Definition 4.2: *A linear BEQ code C_Q is a subspace of $\mathbb{F}_2^n$. Each $\boldsymbol{c} \in C_Q$ is called a codeword of C_Q. The dimension of C_Q is denoted by r.*

Each codeword of C_Q is labeled by a different r-bits sequence $\boldsymbol{u}$. Given a BEQ code C_Q and a source sequence $\boldsymbol{s}'$, a quantization algorithm Q is invoked to find a label $\boldsymbol{u}$ whose codeword $\boldsymbol{c} \in C_Q$ has *no distortion* from $\boldsymbol{s}'$. If such a label is found, it is denoted by $\boldsymbol{u} = Q(\boldsymbol{s}')$, and is considered the compressed vector. Otherwise, a quantization failure is declared, and $Q(\boldsymbol{s}') = Failure$. The reconstruction uses a generator matrix G_Q of C_Q to obtain the codeword $\boldsymbol{c} = \boldsymbol{u}G_Q$.

4.1.3 Reduction from Rewriting to Erasure Quantization

In this subsection we show that the problem of rewriting can be efficiently reduced to that of BEQ. Let C_Q be a linear quantization code, and let H be a parity-check matrix of C_Q.

Construction 4.1: *A rewriting code C_R is constructed as the collection of all cosets of C_Q in $\mathbb{F}_2^n$. A decoding function for C_R is defined by a parity check matrix H of C_Q, such that a vector $\boldsymbol{x} \in \mathbb{F}_2^n$ is decoded into its syndrome:*

$$DEC_H(\boldsymbol{x}) = \boldsymbol{x}H^T. \tag{4.1}$$

Since the dimension of C_Q is r, it has 2^{n-r} cosets. Therefore the rate of C_R is $R_{\text{WOM}} = \frac{n-r}{n}$, implying that $k = n - r$. We define some notation before introducing the reduction algorithm. Let $(H^{-1})^T$ be a left inverse for H^T, meaning that $(H^{-1})^T H^T$ is the $k \times k$ identity matrix. Define a function $BEC : \{0,1\}^n \times \{0,1\}^n \to \{0,1,*\}^n$ as:

$$BEC(\boldsymbol{w}, \boldsymbol{v})_i = \begin{cases} w_i & \text{if } v_i = 0 \\ & \text{if } v_i = 1 \end{cases} \quad , \forall i = 1, ..., n$$

$BEC(\boldsymbol{w}, \boldsymbol{v})$ realizes a binary erasure channel that erases entries in $\boldsymbol{w}$ whose corresponding entries in $\boldsymbol{v}$ equal 1. We are now ready to introduce the encoding algorithm for the rewriting problem.

Algorithm 4.1 $\boldsymbol{x} = ENC(G_Q, \boldsymbol{m}, \boldsymbol{s})$: Encoding for Rewriting

1: $\boldsymbol{z} \leftarrow \boldsymbol{m}(H^{-1})^T$
2: $\boldsymbol{s}' \leftarrow BEC(\boldsymbol{z}, \boldsymbol{s})$
3: $\boldsymbol{u} \leftarrow Q(\boldsymbol{s}')$
4: **if** $\boldsymbol{u}$ = FAILURE **then**
5: **return** FAILURE
6: **else**
7: **return** $\boldsymbol{x} \leftarrow \boldsymbol{u}G_Q + \boldsymbol{z}$
8: **end if**

Theorem 4.1: *Algorithm* 4.1 *either declares a failure or returns a vector* $\boldsymbol{x}$ *such that* $\boldsymbol{x}$ *is rewritable over* $\boldsymbol{s}$ *and* $\boldsymbol{x}H^T = \boldsymbol{m}$.

PROOF: Suppose failure is not declared and $\boldsymbol{x}$ is returned by Algorithm 4.1. We first prove that $\boldsymbol{x}$ is rewritable over $\boldsymbol{s}$. Consider i such that $s_i = 0$. Then it follows from the definition of BEC that $s'_i = z_i$. Remember that $Q(\boldsymbol{s}')$ returns a label $\boldsymbol{u}$ such that $\boldsymbol{c} = \boldsymbol{u}G_Q$ has no-distortion from $\boldsymbol{s}'$. Therefore, $c_i = s'_i = z_i$, and $x_i = c_i + z_i = z_i + z_i = 0 = s'_i$. So $\boldsymbol{x}$ can be written over $\boldsymbol{s}$. To prove the second statement of the theorem, notice that

$$\begin{aligned}\boldsymbol{x}H^T &= (\boldsymbol{u}G_Q + \boldsymbol{z})H^T = \boldsymbol{u}G_Q H^T + \boldsymbol{m}(H^{-1})^T H^T \\ &= \boldsymbol{m}(H^{-1})^T H^T = \boldsymbol{m}.\end{aligned}$$

■

4.2 Rewriting with Message Passing

In this section we discuss how to choose a quantization code C_Q and quantization algorithm Q to obtain a rewriting scheme of good performance. Our approach is to use the iterative quantization scheme of Martinian and Yedidia [58], where C_Q is an LDGM code, and Q is a message-passing algorithm. This approach is particularly relevant for flash memories, since the hardware architecture of message-passing algorithms is well understood and highly optimized in flash controllers.

The algorithm Q can be implemented by a sequential or parallel scheduling, as described in [58, Section 3.4.2]. For concreteness, we consider the sequential algorithm denoted by ERASURE-QUANTIZE in [58]. Since the performance of ERASURE-QUANTIZE depends on the chosen generator matrix, we abuse notation and denote it by $Q(G_Q, \boldsymbol{s}')$. Algorithm $Q(G_Q, \boldsymbol{s}')$ is presented next. We denote $G_Q = (\boldsymbol{g}_1, \ldots, \boldsymbol{g}_n)$ such that $\boldsymbol{g}_j$ is the j-th column of G_Q.

Algorithm 4.2 $\boldsymbol{u} = Q(G_Q, \boldsymbol{s}')$.

$\boldsymbol{v} \leftarrow \boldsymbol{s}'$
while $\exists j$ such that $v_j \neq *$ **do**
  if $\exists i$ such that $\exists! j$ for which $G_Q(i,j) = 1$ and $v_j \neq *$ **then**
    Push (i,j) into the Stack.
    $v_j \leftarrow *$.
  else
    return FAILURE
  end if
end while
$\boldsymbol{u} \leftarrow \boldsymbol{0}_{n-k}$
while Stack is not empty **do**
  Pop (i,j) from the Stack.
  $u_i \leftarrow \boldsymbol{u} \cdot \boldsymbol{g}_j + s'_j$
end while
return $\boldsymbol{u}$

Finally, we need to describe how to choose a generator matrix G_Q that work well together with Algorithm Q. We show next that a matrix G_Q with good rewriting performance can be chosen to be a *parity-check matrix* that performs well in message-passing decoding of erasure channels. This connection follows from the connection between rewriting and quantization, together with a connection between quantization and erasure decoding, shown in [58]. These connections imply that we can use the rich theory and understanding of the design of parity-check matrices in iterative erasure decoding to construct good generating matrices for rewriting schemes. To make the statement precise, we consider the standard iterative erasure-decoding algorithm denoted by ERASURE-DECODE$(H, \boldsymbol{y})$ in [58], where H is an LDPC matrix and $\boldsymbol{y}$ is the output of a binary erasure channel.

Theorem 4.2: *For all* $\boldsymbol{m} \in \mathbb{F}_2^k$ *and* $\boldsymbol{z}', \boldsymbol{s} \in \mathbb{F}_2^n$, *ENC*$(G_Q, \boldsymbol{m}, \boldsymbol{s})$ *fails if and only if* *ERASURE-DECODE*$(G_Q, BEC(\boldsymbol{z}', \boldsymbol{s} + \boldsymbol{1}_n))$ *fails, where* $\boldsymbol{1}_n$ *is the all-one vector of length* n.

Proof: As in Algorithm 4.1, let $\boldsymbol{z} = \boldsymbol{m}(H^{-1})^T$ and $\boldsymbol{s}' = \mathrm{BEC}(\boldsymbol{z}, \boldsymbol{s})$. Now according to Algorithm 4.1, $\mathrm{ENC}(G_Q, \boldsymbol{m}, \boldsymbol{s})$ fails if and only if $Q(G_Q, \boldsymbol{s}')$ fails. According to [58, Theorem 4], $Q(G_Q, \boldsymbol{s}')$ fails if and only if ERASURE-DECODE$(G_Q, \mathrm{BEC}(\boldsymbol{z}', \boldsymbol{s} + \boldsymbol{1}_n))$ fails. This completes the proof. ∎

The running time of the encoding algorithm ENC is analyzed formally in the following theorem.

Theorem 4.3: *The algorithm ENC*$(G_Q, \boldsymbol{m}, \boldsymbol{s})$ *runs in time* $\mathcal{O}(nd)$ *where* n *is the length of* $\boldsymbol{s}$ *and* d *is the maximum degree of the Tanner graph of* G_Q*.*

PROOF: We first show that Step 1 of Algorithm 4.1 runs in time $\mathcal{O}(n)$ if $(H^{-1})^T$ is chosen in the following way. For any C_Q, its parity check matrix H can be made in to systematic form, i.e., $H = (P\ I)$, by row operations and permutation of columns. Then $(H^{-1})^T$ can be chosen as $(\mathbf{0}_{k\times n-k}\ I_k)$, and so $\boldsymbol{z} = \boldsymbol{m}(H^{-1})^T = (\mathbf{0}_{n-k}\ \boldsymbol{m})$.

By [58, Theorem 5], Step 3 of Algorithm 4.1 runs in time $\mathcal{O}(nd)$. By the definition of d, the complexity of Step 7 is also $\mathcal{O}(nd)$. Therefore $\mathcal{O}(nd)$ dominates the computational cost of the algorithm. ∎

Theorems 4.2 and 4.3, together with the analysis and design of irregular LDPC codes that achieve the capacity of the binary erasure channel [65], imply the following capacity-achieving results.

Corollary 4.1: *There exists a sequence of rewriting codes which can be efficiently encoded by Algorithm* 4.1 *and efficiently decoded by Equation* (4.1) *that achieves the capacity of the rewriting model* β*.*

PROOF: Let $\bar{\boldsymbol{s}} = \boldsymbol{s} + \mathbf{1}_n$. Then it follows from Theorem 4.2 that for all G_Q, $\boldsymbol{m} \in \mathbb{F}_2^k$, $\boldsymbol{z}' \in \mathbb{F}_2^n$,

$$\begin{aligned}&\Pr\{ENC(G_Q, \boldsymbol{m}, \boldsymbol{s}) = Failure\} = \\ &\qquad \Pr\{\mathsf{ERASURE\text{-}DECODE}(G_Q, \mathrm{BEC}(\boldsymbol{z}', \bar{\boldsymbol{s}})) = Failure\},\end{aligned}$$

where $\boldsymbol{s}$ is distributed i.i.d. with $\Pr\{s_i =\} = \beta$. The right-hand side is the decoding-failure probability of an LDPC code with parity-check matrix G_Q over a binary erasure channel, using message-passing decoding. The erasure probability of the channel is $1 - \beta$, because $\Pr\{\bar{s}_i = 1\} = 1 - \Pr\{s_i = 1\}$. The capacity of a binary erasure channel with erasure

probability $1-\beta$ is β. This is also the capacity of the rewriting model. In addition, the rate of an LDPC code with parity-check matrix G_Q is equal to the rate of a rewriting code constructed by the cosets of C_Q. It is shown in [65] how to construct a sequence of irregular LDPC codes that achieves the capacity of the binary erasure channel. Such a sequence, used for rewriting codes, achieves the rewriting capacity. ∎

4.2.1 Handling Encoding Failures

The encoding failure event could be dealt with in several ways. A simple solution is to try writing on different invalid pages, if available, or to simply write onto a fresh page, as current flash systems do. If the failure rate is small enough, say below 10^{-3}, the time penalty of rewriting failures would be small. For an alternative solution, we state a reformulation of [58, Theorem 3].

Proposition 4.1: *For all $\boldsymbol{m}, \boldsymbol{m}' \in \mathbb{F}_2^k$ and $\boldsymbol{s} \in \mathbb{F}_2^n$, ENC$(G_Q, \boldsymbol{m}, \boldsymbol{s})$ fails if and only if ENC$(G_Q, \boldsymbol{m}', \boldsymbol{s})$ fails.*

Proof: As in Algorithm 4.1, let $\boldsymbol{z} = \boldsymbol{m}(H^{-1})^T$ and $\boldsymbol{s}' = \mathrm{BEC}(\boldsymbol{z}, \boldsymbol{s})$. Note that $\mathrm{ENC}(G_Q, \boldsymbol{m}, \boldsymbol{s})$ fails if and only if $Q(G_Q, \boldsymbol{s}')$ fails. By Algorithm 4.2, the failure of $Q(G_Q, \boldsymbol{s}')$ is determined only according to the locations of erasures in $\boldsymbol{s}'$, and does not depend on the values of the non-erased entries of $\boldsymbol{s}'$. Since $\boldsymbol{s}' = \mathrm{BEC}(\boldsymbol{z}, \boldsymbol{s})$, the locations of erasures in $\boldsymbol{s}'$ are only determined by the state $\boldsymbol{s}$. This completes the proof. ∎

Proposition 4.1 implies that whether a page is rewritable does not depend on the message to be written. This property suggests that the flash controller can check whether a page is rewritable right after it is being invalidated, without waiting for a message to arrive. An invalid page could be marked as 'unrewritable', such that data would be rewritten only into rewritable pages. This policy would guarantee that the rewriting of a new message always succeeds. However, this policy also implies that the message passing algorithm would run more than once for the rewriting of a page.

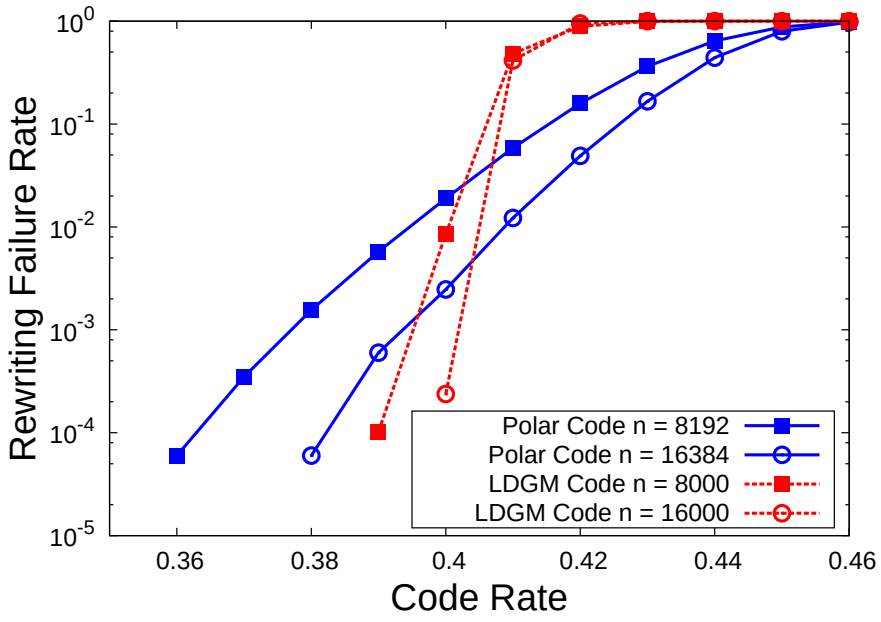

Figure 4.1: Rewriting failure rates of polar and LDGM WOM codes.

4.2.2 Simulation Results

The finite-length performance of our rewriting scheme is evaluated using extensive simulation with the choice of $\beta = 0.5$ and G_Q to be the parity-check matrix of a Mackay code [56]. The rewriting failure rates of our codes with lengths $n = 8000$ and 16000 that are relevant to flash applications are compared with those of the polar WOM codes of lengths 2^{13} and 2^{14} [7]. Fig. 4.1 shows the rewriting failure rates of both codes at different rewriting rate, where each point is calculated from 10^5 experiments. Remember that the capacity of the model is 0.5. The results suggest that our scheme achieves a decent rewriting rate (e.g., 0.39) with low failure rate (e.g., $< 10^{-4}$). Moreover, our codes provide significantly lower failure rates than polar WOM codes when the rewriting rate is smaller, because of the good performance in the waterfall region of message-passing algorithm.

4.3 Error-Correcting Rewriting Codes

The construction of error-correcting rewriting codes is based on a pair of linear codes (C_1, C_Q) that satisfies the condition $C_1 \supseteq C_Q$, meaning that each codeword of C_Q is also a codeword of C_1. Define C_2 to be the dual of C_Q, denoted by $C_2 = C_Q^{\perp}$. A pair of linear codes (C_1, C_2), that satisfies $C_1 \supseteq C_2^{\perp}$ is called *a conjugate code pair*, and it is useful in quantum error correction and cryptography [35]. For the flash memory application, we let C_1 be an error-correction code, while $C_2^{\perp} = C_Q$ is a BEQ code. The main idea in the construction of error-correcting rewriting codes is to label *only* the codewords of C_1, according to their membership in the cosets of C_Q. The construction is defined formally as follows:

Construction 4.2: *For $\boldsymbol{c} \in C_1$, let $\boldsymbol{c} + C_Q$ be the coset of C_Q in C_1 that contains $\boldsymbol{c}$. Then the error-correcting rewriting code is constructed to be the collection of cosets of C_Q in C_1.*

Next we define the matrices $(H^{-1})^T$ and H^T to be used in encoding and decoding. Let G_1 and G_Q be generator matrices of the codes C_1 and C_Q, respectively, such that each row of G_Q is also a row of G_1. Since C_1 contains C_Q, such matrix pair always exists. Define $(H^{-1})^T$ to be constructed by the rows of G_1 that are *not* rows of G_Q. Let H^T be a right inverse of $(H^{-1})^T$.

The encoding is performed according to Algorithm 4.1, with the matrix $(H^{-1})^T$ defined above. Note that in Step 1, $\boldsymbol{z}$ is a codeword of C_1, since each row of $(H^{-1})^T$ is also a row of G_1. In addition, in Step 7, $\boldsymbol{u}G_Q$ is also a codeword of C_1 (unless $Q(G_Q, \boldsymbol{s}')$ fails), since C_Q is contained in C_1. Therefore, $\boldsymbol{x} = \boldsymbol{u}G_Q + \boldsymbol{z}$ is a codeword of C_1. The decoding can begin by the recovery of $\boldsymbol{x}$ from its noisy version, using the decoder of C_1. The message $\boldsymbol{m}$ can then be recovered by the product $\boldsymbol{x}H^T$.

A similar framework was described in [41], which proposed a construction of a repetition code contained in a Hamming code, with a Viterbi encoding algorithm. In this chapter we make the connection to the quantum coding literature, which allows us to construct stronger codes.

4.3.1 Conjugate Codes Construction

good error-correcting code, while $C_2^{\perp}$ is a good LDGM quantization code. Theorem 4.2 implies that C_2 needs to be an LDPC code with a good performance over a binary erasure channel (under message passing decoding).

Constructions of conjugate code pairs in which C_2 is an LDPC code are studied in [34] [40] [73]. Sarvepalli *et al.* [73] constructed a pair of codes such that C_1 is a BCH code and C_2 is a Euclidean geometry LDPC code, which is particularly useful for our purpose. This is because BCH codes are used extensively for error correction in flash memories. Below we first briefly review the construction of Euclidean geometry LDPC codes and then discuss the application of the results in [73] to our settings.

Denote by $\mathrm{EG}(m, p^s)$ the Euclidean finite geometry over $\mathbb{F}_{p^s}$ consisting of p^{ms} points. Note that this geometry is equivalent to the vector space $\mathbb{F}_{p^s}^m$. A μ-dimensional subspace of $\mathbb{F}_{p^s}^m$ or its coset is called a *μ-flat*. Let J be the number of μ-flats that do not contain the origin, and let $\alpha_1, ...\alpha_{p^{sm}-1}$ be the points of $\mathrm{EG}(m, p^s)$ excluding the origin. Construct a $J \times p^{sm} - 1$ matrix H_{EG} in the way that its (i, j)-th entry equals 1 if the i-th μ-flat contains α_j, and equals 0 otherwise. H_{EG} is the parity check matrix of the (Type-I) Euclidean geometry LDPC code $C_{EG}(m, \mu, s, p)$. $C_{EG}(m, \mu, s, p)$ is a cyclic code, and by analyzing the roots of its generator polynomial the following result is obtained [73]:

Proposition 4.2: *$C_{EG}^{\perp}(m, \mu, s, p)$ is contained in a BCH code of design distance $\delta = p^{\mu s} - 1$.*

Hence we may choose C_2 to be $C_{EG}(m, \mu, s, p)$ and C_1 to be a BCH code with distance equal to or smaller than δ. Some possible code constructions are shown in Table 4.1. Their encoding performance, with respect to the probability β that a cell in the state is writable, is shown in Fig. 4.2. Note from Fig. 4.2 that a code with smaller rewriting rate achieves a fixed failure rate at a smaller value of β. In particular, the codes corresponding to the top three rows of Table 4.1 achieve very small failure rate at $\beta = 0.5$, the point of practical interest. These results also show that the slope of the figures becomes sharper when the length of the codes increases, as expected. Out of the three codes that can be rewritten with $\beta = 0.5$,

Table 4.1: Error-correcting Rewriting Codes Constructed from Pairs of Conjugate BCH and EG-LDPC Codes.

(m, μ, s, p)	$C_1[n, k, \delta]$	$C_2[n, k]$	Rewriting Rate
(4,1,2,2)	[255,247,3]	[255,21]	0.0510
(3,1,2,2)	[65,57,3]	[65,13]	0.1111
(3,1,3,2)	[511,484,7]	[511,139]	0.2192
(3,1,4,2)	[4095,4011,15]	[4095,1377]	0.3158

$C_{EG}(3, 1, 3, 2)$ poses the best rate and error-correction capability.

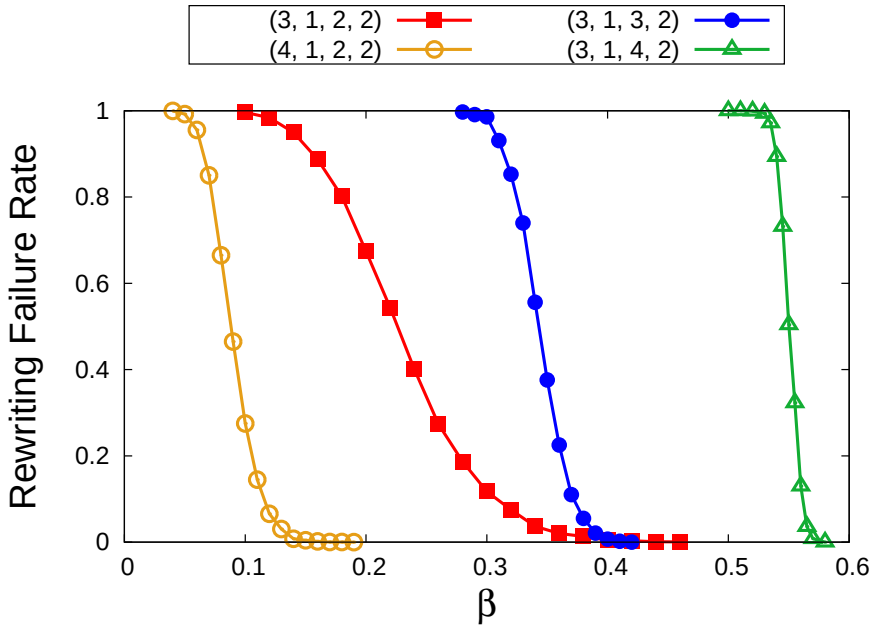

Figure 4.2: Encoding performance of the codes in Table 4.1.

In this section we present two alternative approaches to combine rewriting codes with error correction.

4.3.2 Concatenated Codes

In this scheme, we concatenate an LDGM rewriting code with a systematic error-correcting code. The outer code is an LDGM rewriting code without error-correction capability, as in Section 4.2. The systematic ECC is used as the inner code. The concatenated scheme is

used in the second write. The scheme requires the first write to *reserve* some bits to store the redundancy of the ECC in the second write.

In the second write, the encoder begins by finding a vector $\boldsymbol{x}$ that can be written over the current state. After $\boldsymbol{x}$ is written, the systematic ECC calculates the redundancy bits required to protect $\boldsymbol{x}$ from errors. The redundancy bits are then written into the reserved cells. The decoding of the second write begins by recovering $\boldsymbol{x}$ using the systematic ECC and its redundancy bits. After $\boldsymbol{x}$ is recovered, the decoder of the rewriting code recovers the stored message from $\boldsymbol{x}$.

We note that reserving bits for the second write have a negative effect on the performance of the system, since it reduces the total amount of information that can be stored in the memory on a given time. Therefore, the next subsection extends the concatenation scheme using a chaining technique, with the aim of reducing the number of bits required to be reserved for the second write.

4.3.3 Code Chaining

The chaining approach is inspired by a similar construction in polar coding [63]. The idea is to chain several code blocks of short length. In the following we use a specific example to demonstrate the idea. We use a BCH code for error correction, since its performance can be easily calculated. We note, however, that LDPC codes may be used in practice, such that the circuit modules may be shared with the rewriting code, to reduce the required area. The performance of LDPC code in the considered parameters is similar to that of BCH codes.

A typical BCH code used in flash memory has the parameters $[8191, 7671, 81]$, where the length is 8191, the dimension is 7671, and the minimum distance is 81. If this code is used in a concatenated scheme for the second write, the first write needs to reserve $8191-7671 = 520$ bits for redundancy.

To reduce the amount of required reserved bits, we consider the chaining of 8 systematic BCH codes with the parameters $[1023, 863, 33]$. The encoding is performed sequentially, beginning with the rewriting encoding that finds a vector $\boldsymbol{x}_1$ of 863 bits. The vector $\boldsymbol{x}_1$

represents a message $\boldsymbol{m}_1$ of 310 bits, according to an $[863, 310]$-LDGM rewriting code. Once $\boldsymbol{x}_1$ is found, the BCH encoder finds $1023-863 = 160$ redundancy bits to protect $\boldsymbol{x}_1$, as in the concatenated scheme. The encoder then "chains" the redundancy bits forward, by encoding them, together with 150 new information bits, into another block of 863 bits, using the $[863, 310]$-LDGM code. Let $\boldsymbol{m}_2$ denote the vector of 310 bits encoded into the second block. $\boldsymbol{m}_2$ contains the 160 redundancy bits of $\boldsymbol{x}_1$, together with the additional 150 information bits. Note that once $\boldsymbol{m}_2$ is decoded, the redundancy bits of $\boldsymbol{x}_1$ are available, allowing the recovery $\boldsymbol{x}_1$, and then $\boldsymbol{m}_1$. The encoding continues in this fashion 8 times, to write over a total of 8 blocks, each containing 863 cells. The 160 redundant bits used to protect $\boldsymbol{x}_8$ are stored in the reserved cells. The decoding is done in the reverse order, where each decoded vector contains the redundancy bits of the previous block.

4.3.4 Comparison

We compare the different error-correction approaches, and discuss their trade-offs. The first code in the comparison is a conjugate code pair, described in Section 4.3. We use a conjugation of a $[511, 484, 7]$-BCH code containing a $[511, 372]$-LDGM code, dual to the $(3, 1, 3, 2)$-Euclidean geometry LDPC code in Table 4.1. The second code in the comparison is a concatenation of an outer $[7671, 2915]$-LDGM Mackay rewriting code with an inner $[8191, 7671, 81]$-BCH code. The third code is a chaining of 8 blocks of $[863, 310]$-LDGM Mackay codes, each concatenated with a $[1023, 863, 33]$-BCH code. We compare the decoding BER P_D, the fraction α of bits required to be reserved, and the rewriting rate R_{WOM} of the codes. The encoding failure rate of each of the three codes for $\beta = 0.5$ is below 10^{-3}. P_D is estimated with a standard flash memory assumption of a raw BER of 1.3×10^{-3}. To achieve a comparable code length, the conjugated code is assumed to be used 16 times in parallel, with a total length of $511 \times 16 = 8176$. The comparison is summarized in Table 4.2.

Flash systems require P_D below 10^{-15}. We see in Table 4.2 that conjugated code still do not satisfy the reliability requirement. We also see that concatenated codes that satisfy the reliability requirement need a large fraction of reserved space. The chained code reduces the

Table 4.2: Error-correcting rewriting codes of length ≈ 8200.

Code	P_D	α	R_{WOM}
Conjugated	10^{-5}	0%	0.21
Concatenated	10^{-16}	6.3%	0.35
Chained	10^{-16}	2%	0.19

fraction of reserved space to 2%, with a rate penalty in the second write.

4.4 Summary

In this chapter we proposed WOM schemes based on the cosets of LDGM codes, together with a message-passing algorithm. We gave several arguments and simulation results that suggest that the scheme could be useful for practical implementation and application.

Chapter 5

Rewriting with Polar Codes

The material in this chapter was presented in part in [21]. In this chapter we present a new construction of polar WOM codes. We first discuss the relation of the result to previous work.

5.1 Relation to Previous Work

The study of channel coding with an informed encoder was initiated by Kusnetsov and Tsybakov [51], with the channel capacity derived by Gelfand and Pinsker [28]. The informed encoding technique of Gelfand and Pinsker was used earlier by Marton to establish an inner bound for the capacity region of broadcast channels [59]. Low-complexity capacity-achieving codes were first proposed for continuous channels, using lattice codes [90]. In discrete channels, the first low-complexity capacity-achieving scheme was proposed using polar codes, for the symmetric special case of information embedding [50, Section VIII.B]. A modification of this scheme for the WOM model was proposed in [7]. An additional capacity-achieving WOM scheme, based on randomness extractors, was also proposed recently [25].

Our work is concerned with a setup that is similar to those considered in [7, 25]. An important contribution of the current work compared to [7, 25] is that our scheme achieves the capacity of a rewriting model that also includes noise, while the schemes in [7,25] address only the noiseless case. Indeed, error correction is a crucial capability in flash memory

systems. Our low-complexity achievability of the noisy capacity is done using a multicoding technique. Compared with [25], the current work allows an input cost constraint, which is important in rewriting models for maximizing the sum of the code rates over multiple rewriting rounds. Compared with [7], the current work also improves by removing the requirement for shared randomness between the encoder and the decoder, which limits the practical coding performance. The removal of the shared randomness is done by the use of non-linear polar codes. An additional coding scheme was proposed during the writing of this thesis, and which also does not require shared randomness [55]. However, the scheme in [55] considers only the noiseless case, and it is in fact a special case of the scheme in the current chapter.

Polar coding for channels with informed encoders was implicitly studied recently in the context of broadcast channels, as the Marton coding scheme for broadcast communication contains an informed encoding instance as an ingredient. In fact, a multicoding technique similar to the one presented in this chapter was recently presented for broadcast channels, in [29]. While we were unaware of the result of [29] and developed the scheme independently, this chapter also has three contributions that were not shown in [29]. First, by using the modified scheme of non-linear polar codes, we reduce the storage requirement from an exponential function in the block length to a linear function. Secondly, we connect the scheme to the application of data storage and flash memory rewriting, which was not considered in the previous work. And thirdly, the analysis in [29] holds only for channels whose capacity-achieving distribution forms a certain degraded structure. In this chapter we consider a specific noisy rewriting model, whose capacity-achieving distribution forms the required degraded structure, and by that we show that the scheme achieves the capacity of the considered flash-memory model.

Another recent paper on polar coding for broadcast channels was published recently by Mondelli et. al. [63]. That paper used the chaining method (as in Subsection 4.3.3), which allows one to bypass the degraded structure requirement. In this chapter we connect the chaining method to the flash-memory rewriting application and to our new non-linear polar

coding scheme, and apply it to our proposed multicoding scheme. This allows for a linear storage requirement, together with the achievability of the informed encoder capacity and Marton's inner bound, eliminating the degraded structure requirement. Finally, we show an important instance of the chaining scheme for a specific flash-memory model, and explain the applicability of this instance in flash-memory systems.

A special case of the proposed WOM construction is that of point-to-point channel coding. The proposed construction also contributes to the literature on point-to-point channel coding, in the case that the channel is asymmetric. Several polar coding schemes for asymmetric channels were proposed recently, including a pre-mapping using Gallager's scheme [26, p. 208] and a concatenation of two polar codes [80]. A more direct approach was proposed in [37], and also considered in [62]. The scheme in [37] achieves the capacity of asymmetric channels using non-linear polar codes, but it uses large Boolean functions that require storage space that is exponential in the block length. We propose a modification for this scheme that removes the requirement for the Boolean functions, and thus reduces the storage requirement of the encoding and decoding tasks to a linear function of the block length.

We start by describing the special case of asymmetric channels in Section 5.2. We then move to the WOM case in Section 5.3, which proposes a polar multicoding scheme for channels with informed encoders, including two special cases of noisy WOM.

5.2 Asymmetric Point-to-Point Channels

Notation: For positive integers $m \leq n$, let $[m:n]$ denote the set $\{m, m+1, \ldots, n\}$, and let $[n]$ denote the set $[1:n]$. Given a subset $\mathcal{A}$ of $[n]$, let $\mathcal{A}^c$ denote the complement of $\mathcal{A}$ with respect to $[n]$, where n is clear from the context. Let $x_{[n]}$ denote a vector of length n, and let $x_{\mathcal{A}}$ denote a vector of length $|\mathcal{A}|$ obtained from $x_{[n]}$ by deleting the elements with indices in $\mathcal{A}^c$.

Throughout this section we consider only channels with binary input alphabets, since the literature on polar codes with non-binary codeword symbols is relatively immature. However,

the results of this section can be extended to non-binary alphabets without much difficulty using the methods described in [64, 66, 72, 74–76]. The main idea of polar coding is to take advantage of the polarization effect of the Hadamard transform on the entropies of random vectors. Consider a binary-input memoryless channel model with an input random variable (RV) $X \in \{0, 1\}$, an output RV $Y \in \mathcal{Y}$ and a pair of conditional probability mass functions (pmfs) $p_{Y|X}(y|0), p_{Y|X}(y|1)$ on $\mathcal{Y}$. Let n be a power of 2 that denotes the number of channel uses, also referred to as the block length. The channel capacity is the tightest upper bound on the code rate in which the probability of decoding error can be made as small as desirable for a large enough block length. The channel capacity is given by the mutual information of X and Y.

Theorem 5.1: ***(Channel Coding Theorem)*** *[13, Chapter 7] The capacity of the discrete memoryless channel $p(y|x)$ is given by*

$$C = \max_{p(x)} I(X; Y).$$

The Hadamard transform is a multiplication of the random vector $X_{[n]}$ over the field of cardinality 2 with the matrix $G_n = G^{\otimes \log_2 n}$, where $G = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ and $\otimes$ denotes the Kronecker power. In other words, G_n can be described recursively for $n \geq 4$ by the block matrix

$$G_n = \begin{pmatrix} G_{n/2} & 0 \\ G_{n/2} & G_{n/2} \end{pmatrix}.$$

The matrix G_n transforms $X_{[n]}$ into a random vector $U_{[n]} = X_{[n]}G_n$, such that the conditional entropy $H(U_i|U_{[i-1]}, Y_{[n]})$ is *polarized*. That means that for a fraction of close to $H(X|Y)$ of the indices $i \in [n]$, the conditional entropy $H(U_i|U_{[i-1]}, Y_{[n]})$ is close to 1, and for almost all the rest of the indices, $H(U_i|U_{[i-1]}, Y_{[n]})$ is close to 0. This result was shown by Arikan in [1, 2].

Theorem 5.2: ***(Polarization Theorem)*** *[2, Theorem 1] Let $n, U_{[n]}, X_{[n]}, Y_{[n]}$ be defined*

as above. For any $\delta \in (0,1)$*, let*

$$H_{X|Y} \triangleq \left\{ i \in [n] : H(U_i|U_{[i-1]}, Y_{[n]}) \in (1-\delta, 1) \right\},$$

and

$$L_{X|Y} \triangleq \left\{ i \in [n] : H(U_i|U_{[i-1]}, Y_{[n]}) \in (0, \delta) \right\}.$$

Then

$$\lim_{n\to\infty} |H_{X|Y}|/n = H(X|Y) \quad \textit{and} \quad \lim_{n\to\infty} |L_{X|Y}|/n = 1 - H(X|Y).$$

Note that $H(X|Y)$ denotes a conditional entropy, while $H_{X|Y}$ denotes a subset of $[n]$. It is also shown in [1] that the transformation G_n is invertible with $G_n^{-1} = G_n$, implying $X_{[n]} = U_{[n]}G_n$. This polarization effect can be used quite simply for the design of a coding scheme that achieves the capacity of symmetric channels with a running time that is polynomial in the block length. The capacity of symmetric channels is achieved by a uniform distribution on the input alphabet, i.e., $p(x) = 1/2$ [13, Theorem 7.2.1]. Since the input alphabet in this chapter is binary, the capacity-achieving distribution gives $H(X) = 1$, and therefore we have

$$\lim_{n\to\infty} (1/n)|L_{X|Y}| = 1 - H(X|Y) = H(X) - H(X|Y) = I(X;Y) = C. \tag{5.1}$$

Furthermore, for each index in $L_{X|Y}$, the conditional probability $p(u_i|u_{[i-1]}, y_{[n]})$ must be close to either 0 or 1 (since the conditional entropy is small by the definition of the set $L_{X|Y}$). It follows that the RV U_i can be estimated reliably given $u_{[i-1]}$ and $y_{[n]}$. This fact motivates the capacity-achieving coding scheme that follows. The encoder creates a vector $u_{[n]}$ by assigning the subvector $U_{L_{X|Y}}$ with the source message, and the subvector $U_{L^c_{X|Y}}$ with uniformly distributed random bits that are shared with the decoder. The randomness sharing is useful for the analysis, but is in fact unnecessary for using the scheme (the proof of this fact is described in [1, Section VI]). The set $U_{L^c_{X|Y}}$ is called the *frozen set*. Equation (5.1) implies that this coding rate approaches the channel capacity. The decoding is performed iteratively, from index 1 up to n. In each iteration, the decoder estimates the bit u_i using the shared

information or using a maximum likelihood estimation, according to the set membership of the iteration. The estimations of the bits u_i for which i is in $L^c_{X|Y}$ are always successful, since these bits were known to the decoder in advance. The rest of the bits are estimated correctly with high probability, leading to a successful decoding of the entire message with high probability.

However, this reasoning does not translate directly to asymmetric channels. Remember that the capacity-achieving input distribution of asymmetric channels is in general not uniform (see, for example, [32]), i.e., $p_X(1) \neq 1/2$. Since the Hadamard transform is bijective, it follows that the capacity-achieving distribution of the polarized vector $U_{[n]}$ is non uniform as well. The problem with this fact is that assigning uniform bits of message or shared randomness changes the distribution of $U_{[n]}$, and consequentially also changes the conditional entropies $H(U_i|U_{[i-1]}, Y_{[n]})$. To manage this situation, our approach is to make sure that the change in the distribution of $U_{[n]}$ is kept to be minor, and thus its effect on the probability of decoding error is also minor. To do this, we consider the conditional entropies $H(U_i|U_{[i-1]})$, for $i \in [n]$. Since the polarization happens regardless of the channel model, we can consider a channel for which the output Y is a deterministic variable, and conclude by Theorem 5.2 that the entropies $H(U_i|U_{[i-1]})$ also polarize. For this polarization, a fraction of $H(X)$ of the indices admit a high $H(U_i|U_{[i-1]})$. To ensure a minor change in the distribution of $U_{[n]}$, we restrict the assignments of uniform bits of message and shared randomness to the indices with high $H(U_i|U_{[i-1]})$.

The insight of the last paragraph motivates a modified coding scheme. The locations

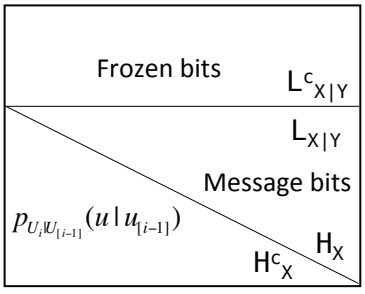

Figure 5.1: Encoding the vector $u_{[n]}$.

with high entropy $H(U_i|U_{[i-1]})$ are assigned with uniformly distributed bits, while the rest of the locations are assigned with the pmf $p(u_i|u_{[i-1]})$. Note that $p(u_{[n]}, x_{[n]})$ and $H(U_{[n]})$ refer to the capacity-achieving distribution of the channel, which does not equal the distribution that the encoding process induces. Similar to the notation of Theorem 5.2, we denote the set of indices with high entropy $H(U_i|U_{[i-1]})$ by H_X. To achieve a reliable decoding, we place the message bits in the indices of H_X that can be decoded reliably, meaning that their entropies $H(U_i|U_{[i-1]}, Y_{[n]})$ are low. So we say that we place the message bits in the intersection $H_X \cap L_{X|Y}$. The locations whose indices are not in $L_{X|Y}$ must be known by the decoder in advance for a reliable decoding. Previous work suggested sharing random Boolean functions between the encoder and the decoder, drawn according to the pmf $p(u_i|u_{[i-1]})$, and to assign the indices in $(H_X \cap L_{X|Y})^c = H_X^c \cup L_{X|Y}^c$ according to these functions [29, 37]. However, we note that the storage required for those Boolean functions is exponential in n, and therefore we propose an efficient alternative.

To avoid the Boolean function, we divide the complement of $H_X \cap L_{X|Y}$ into three disjoint sets. First, the indices in the intersection $H_X \cap L_{X|Y}^c$ are assigned with uniformly distributed random bits that are shared between the encoder and the decoder. As in the symmetric case, this randomness sharing will in fact not be necessary, and a deterministic frozen vector could be shared instead. The rest of the bits of $U_{[n]}$ (those in the set H_X^c) are assigned *randomly* to a value u with probability $p_{U_i|U_{[i-1]}}(u|u_{[i-1]})$ (where $p_{U_i|U_{[i-1]}}$ is calculated according to the pmf $p_{U_{[n]},X_{[n]},Y_{[n]}}$, the capacity-achieving distribution of the channel). The indices in $H_X^c \cap L_{X|Y}$ could be decoded reliably, but not those in $H_X^c \cap L_{X|Y}^c$. Fortunately, the set $H_X^c \cap L_{X|Y}^c$ can be shown to be small (as we will show later), and thus we could transmit those locations separately with a vanishing effect on the code rate. The encoding of the vector $u_{[n]}$ is illustrated in Figure 5.1.

The see the intuition of why the code rate approaches the channel capacity, notice that the source message is placed in the indices in the intersection $H_X \cap L_{X|Y}$. The asymptotic

fraction of this intersection can be derived as following:

$$|H_X \cap L_{X|Y}|/n = 1 - |H_X^c \cup L_{X|Y}^c|/n = 1 - |H_X^c|/n - |L_{X|Y}^c|/n + |H_X^c \cap L_{X|Y}^c|/n. \quad (5.2)$$

The Polarization Theorem (Theorem 5.2) implies that $|H_X^c|/n \to 1 - H(X)$ and $|L_{X|Y}^c|/n \to H(X|Y)$. Since the fraction $|H_X^c \cap L_{X|Y}^c|$ vanishes for large n, we get that the asymptotic rate is $|H_X \cap L_{X|Y}|/n \to H(X) - H(X|Y) = I(X;Y)$, achieving the channel capacity.

For a more precise definition of the scheme, we use the so called Bhattacharyya parameter in the selection of subsets of $U_{[n]}$, instead of the conditional entropy. The Bhattacharyya parameters are polarized in a similar manner as the entropies, and are more useful for bounding the probability of decoding error. For a discrete RV Y and a Bernoulli RV X, the Bhattacharyya parameter is defined by

$$Z(X|Y) \triangleq 2 \sum_y \sqrt{p_{X,Y}(0,y) p_{X,Y}(1,y)}. \quad (5.3)$$

Note that most of the polar coding literature uses a slightly different definition of the Bhattacharyya parameter that coincides with Equation (5.3) when the RV X is distributed uniformly. We use the following relations between the Bhattacharyya parameter and the conditional entropy.

Proposition 5.1: *([2, Proposition 2])*

$$(Z(X|Y))^2 \leq H(X|Y), \quad (5.4)$$

$$H(X|Y) \leq \log_2(1 + Z(X|Y)) \leq Z(X|Y). \quad (5.5)$$

We now define the set of high and low Bhattacharyya parameters, and work with them

instead of the sets $H_{X|Y}$ and $L_{X|Y}$. For $\delta \in (0,1)$, define

$$\mathcal{H}_{X|Y} \triangleq \left\{ i \in [n] : Z(U_i|U_{[i-1]}, Y_{[n]}) \geq 1 - 2^{-n^{1/2-\delta}} \right\},$$
$$\mathcal{L}_{X|Y} \triangleq \left\{ i \in [n] : Z(U_i|U_{[i-1]}, Y_{[n]}) \leq 2^{-n^{1/2-\delta}} \right\}.$$

As before, we define the sets $\mathcal{H}_X$ and $\mathcal{L}_X$ for the parameter $Z(U_i|U_{[i-1]})$ by letting $Y_{[n]}$ be a deterministic vector. Using Proposition 5.1, it is shown in [37, combining Proposition 2 with Theorem 2] that Theorem 5.2 holds also if we replace the sets $H_{X|Y}$ and $L_{X|Y}$ with the sets $\mathcal{H}_{X|Y}$ and $\mathcal{L}_{X|Y}$. That is, we have

$$\lim_{n\to\infty} |\mathcal{H}_{X|Y}|/n = H(X|Y) \quad \text{and} \quad \lim_{n\to\infty} |\mathcal{L}_{X|Y}|/n = 1 - H(X|Y). \tag{5.6}$$

We now define our coding scheme formally. Let $m_{[|\mathcal{H}_X \cap \mathcal{L}_{X|Y}|]} \in \{0,1\}^{|\mathcal{H}_X \cap \mathcal{L}_{X|Y}|}$ be the realization of a uniformly distributed source message, and $f_{[|\mathcal{H}_X \cap \mathcal{L}^c_{X|Y}|]} \in \{0,1\}^{|\mathcal{H}_X \cap \mathcal{L}^c_{X|Y}|}$ be a deterministic frozen vector known to both the encoder and the decoder. We discuss how to find a good frozen vector in Appendix 5.6.3. For a subset $\mathcal{A} \subseteq [n]$ and an index $i \in \mathcal{A}$, we use a function $r(i, \mathcal{A})$ to denote the *rank* of i in an ordered list of the elements of $\mathcal{A}$. The probabilities $p_{U_i|U_{[i-1]}}(u|u_{[i-1]})$ and $p_{U_i|U_{[i-1]},Y^n}(u|u_{[i-1]}, y_{[n]})$ can be calculated efficiently by a recursive method described in [37, Section III.B].

Construction 5.1:

Encoding

Input*: a message* $m_{[|\mathcal{H}_X \cap \mathcal{L}_{X|Y}|]} \in \{0,1\}^{|\mathcal{H}_X \cap \mathcal{L}_{X|Y}|}$.

Output: *a codeword* $x_{[n]} \in \{0,1\}^n$.

1. *For* i *from* 1 *to* n*, successively, set*

$$u_i = \begin{cases} u \in \{0,1\} \quad \textit{with probability } p_{U_i|U_{[i-1]}}(u|u_{[i-1]}) & \textit{if } i \in \mathcal{H}^c_X \\ m_{r(i,\mathcal{H}_X \cap \mathcal{L}_{X|Y})} & \textit{if } i \in \mathcal{H}_X \cap \mathcal{L}_{X|Y} \\ f_{r(i,\mathcal{H}_X \cap \mathcal{L}^c_{X|Y})} & \textit{if } i \in \mathcal{H}_X \cap \mathcal{L}^c_{X|Y}. \end{cases}$$

2. *Transmit the codeword* $x_{[n]} = u_{[n]}G_n$.

3. *Transmit the vector* $u_{\mathcal{H}_X^c \cap \mathcal{L}_{X|Y}^c}$ *separately using a linear, non-capacity-achieving polar code with a uniform input distribution (as in [1]). In practice, other error-correcting codes could be used for this vector as well.*

Decoding

Input*: a noisy vector* $y_{[n]} \in \{0,1\}^n$.

Output: *a message estimation* $\hat{m}_{[|\mathcal{H}_X \cap \mathcal{L}_{X|Y}|]} \in \{0,1\}^{|\mathcal{H}_X \cap \mathcal{L}_{X|Y}|}$.

1. *Estimate the vector* $u_{\mathcal{H}_X^c \cap \mathcal{L}_{X|Y}^c}$ *by* $\hat{u}_{\mathcal{H}_X^c \cap \mathcal{L}_{X|Y}^c}$.

2. *For* i *from 1 to* n*, set*

$$\hat{u}_i = \begin{cases} \underset{u \in \{0,1\}}{\arg\max}\, p_{U_i|U_{[i-1]},Y_{[n]}}(u|u_{[i-1]}, y_{[n]}) & \text{if } i \in \mathcal{L}_{X|Y} \\ \hat{u}_{r(i,\mathcal{H}_X^c \cap \mathcal{L}_{X|Y}^c)} & \text{if } i \in \mathcal{H}_X^c \cap \mathcal{L}_{X|Y}^c \\ f_{r(i,\mathcal{H}_X \cap \mathcal{L}_{X|Y}^c)} & \text{if } i \in \mathcal{H}_X \cap \mathcal{L}_{X|Y}^c. \end{cases}$$

3. *Return the estimated message* $\hat{m}_{[|\mathcal{H}_X \cap \mathcal{L}_{X|Y}|]} = \hat{u}_{\mathcal{H}_X \cap \mathcal{L}_{X|Y}}$.

We say that a sequence of coding schemes achieves the channel capacity if the probability of decoding error vanishes with the block length for any rate below the capacity.

Theorem 5.3: *Construction* 5.1 *achieves the channel capacity (Theorem* 5.1*) with a computational complexity of* $O(n \log n)$ *and a probability of decoding error of* $2^{-n^{1/2-\delta}}$ *for any* $\delta > 0$ *and large enough* n.

In the next section we show a generalized construction and prove its capacity-achieving property. Theorem 5.3 thus will follow as a corollary of the more general Theorem 5.7.

5.3 Channels with Non-Causal Encoder State Information

In this section we generalize Construction 5.1 to the availability of channel state information at the encoder. We consider mainly the application of rewriting in flash memories (WOM), and present two special cases of the channel model for this application. We model the memory cells as a channel with a discrete *state*, and we also assume that the state is memoryless, meaning that the states of different cells are distributed independently. We assume that the state of the entire n cells is available to the writer prior to the beginning of the writing process. In communication terminology this kind of state availability is refereed to as "non causal". We note that this setting is also useful in the so called *Marton-coding* method for communication over broadcast channels. Therefore, the multicoding schemes that will follow serve as an additional contribution in this important setting.

We represent the channel state as a Bernoulli random variable S with parameter β, which equals the probability $p(S = 1)$. A cell of state 1 can only be written with the value 1. Note that, intuitively, when β is high, the capacity of the memory is small, since only a few cells are available for modification in the writing process, and thus only a small amount of information could be stored. This also means that the choice of codebook has a crucial effect on the capacity of the memory in *future writes*. A codebook that contains many codewords of high Hamming weight (number of 1's in the codeword) would make the parameter β of future writes high, and thus the capacity of the future writes would be low. However, forcing the expected Hamming weight of the codebook to be low would reduce the capacity of the current write. To settle this trade-off, previous work suggested to optimize the sum of the code rates over multiple writes. It was shown that in many cases, constraints on the codebook Hamming weight (henceforth just weight) strictly increase the sum rate (see, for example, [36]). Therefore, we consider an input cost constraint in the model.

The most general model that we consider is a discrete memoryless channel (DMC) with a discrete memoryless (DM) state and an input cost constraint, where the state information

is available non-causally at the encoder. The channel input, state, and output are denoted by x, s, and y, respectively, and their respective finite alphabets are denoted by $\mathcal{X}, \mathcal{S}$, and $\mathcal{Y}$. The random variables are denoted by X, S, and Y, and the random vectors by $X_{[n]}, S_{[n]}$, and $Y_{[n]}$, where n is the block length. The state is distributed according to the pmf $p(s)$, and the conditional pmfs of the channel are denoted by $p(y|x, s)$. The input cost function is denoted by $b(x)$, and the input cost constraint is

$$\sum_{i=1}^{n} \mathsf{E}[b(X_i)] \leq nB,$$

where B is a real number representing the normalized constraint. The channel capacity with an informed encoder and an input cost constraint is given by an extension of the Gelfand-Pinsker Theorem[1].

Theorem 5.4: ***(Gelfand-Pinsker Theorem with Cost Constraint)*** *[16, Equation (7.7) on p. 186] The capacity of a DMC with a DM state $p(y|x, s)p(s)$ and an input cost constraint B when the state information is available non causally only at the encoder is*

$$C = \max_{p(v|s), x(v,s): \mathsf{E}(b(X)) \leq B} (I(V;Y) - I(V;S)), \tag{5.7}$$

where V is an auxiliary random variable with a finite alphabet $\mathcal{V}$,
$|\mathcal{V}| \leq \min\{|\mathcal{X} \cdot \mathcal{S}, \mathcal{Y} + \mathcal{S} - 1\}$.

The main coding scheme that we present in this section achieves the capacity in Theorem 5.4. The proof of Theorem 5.4 considers a virtual channel model, in which the RV V is the channel input and Y is the channel output. Similar to the previous section, we limit the treatment to the case in which the RV V is binary. In flash memory, this case would correspond to a single-level cell (SLC) type of memory. As mentioned in Section 5.2, an extension of the scheme to a non-binary case is not difficult. The non-binary case is useful for flash

[1]The cost constraint is defined slightly differently in this reference, but the capacity is not affected by this change.

memories in which each cell stores 2 or more bits of information. Such memories are called Multi-Level Cell (MLC). We also mention that the limitation to binary random variables does not apply on the channel output Y. Therefore, the cell voltage in flash memory could be read more accurately at the decoder to increase the coding performance, similarly to the soft decoding method that is used in flash memories with LDPC codes. Another practical remark is that the binary-input model can be used in MLC memories by coding separately on the MSB and the LSB of the cells, as is in fact the coding method in current MLC flash systems.

The scheme that achieves the capacity of Theorem 5.4 is called Construction 5.3, and it will be described in Subsection 5.3.3. The capacity achieving result is summarized in the following theorem, which will be proven in Subsection 5.3.3.

Theorem 5.5: *Construction* 5.3 *achieves the capacity of the Gelfand-Pinsker Theorem with Cost Constraint (Theorem* 5.4*) with a computational complexity of* $O(n \log n)$ *and a probability of decoding error of* $2^{-n^{1/2-\delta}}$ *for any* $\delta > 0$ *and large enough* n*.*

Note that the setting of Theorem 5.5 is a generalization of the asymmetric channel-coding setting of Theorem 5.3, and therefore Construction 5.3 and Theorem 5.5 are in fact a generalization of Construction 5.1 and Theorem 5.3. Before we describe the code construction, we first show in Subsection 5.3.1 two special cases of the Gelfand-Pinsker model that are useful for the rewriting of flash memories. Afterwards, in subsections 5.3.2 and 5.3.3, we will show two versions of the construction that correspond to generalizations of the two special cases.

5.3.1 Special Cases

We start with a special case that is quite a natural model for flash memory rewriting.

Example 5.1: *Let the sets* $\mathcal{X}, \mathcal{S}$*, and* $\mathcal{Y}$ *be all equal to* $\{0, 1\}$*, and let the state pmf be* $p_S(1) = \beta$*. This model corresponds to a single level cell flash memory. We describe the cell behaviour after a bit* x *is attempted to be written. When* $s = 0$*, the cell behaves as*

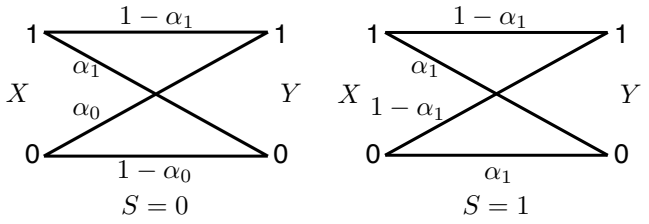

Figure 5.2: Example 5.1: A binary noisy WOM model.

a binary asymmetric channel with input x, since the call state does not interfere with the writing attempt. When $s = 1$, the cell behaves as if a value of 1 was attempted to be written, regardless of the actual value x attempted. However, an error might still occur, during the writing process or anytime afterwards (for example, due to charge leakage). Thus, we can say that when $s = 1$, the cell behaves as a binary asymmetric channel with input 1. Formally, the channel pmfs are given by

$$p_{Y|XS}(1|x,s) = \begin{cases} \alpha_0 & \text{if } (x,s) = (0,0) \\ 1-\alpha_1 & \text{if } (x,s) = (0,1) \\ 1-\alpha_1 & \text{if } (x,s) = (1,0) \\ 1-\alpha_1 & \text{if } (x,s) = (1,1) \end{cases} \tag{5.8}$$

The error model is also presented in Figure 5.2*. The cost constraint is given by $b(x_i) = x_i$, since it is desirable to limit the amount of cells written to a value of 1.*

Our coding-scheme construction for the setting of Theorem 5.4 is based on a more limited construction, which serves as a building block. We will start by describing the limited construction, and then show how to extend it for the model of Theorem 5.4. We will prove that the limited construction achieves the capacity of channels whose capacity-achieving distribution forms a certain stochastically degraded structure. We first recall the definition of stochastically degraded channels.

Definition 5.1: *[16, p. 112] A discrete memoryless channel (DMC) $W_1 : \{0,1\} \to \mathcal{Y}_1$ is* stochastically degraded *(or simply degraded) with respect to a DMC $W_2 : \{0,1\} \to \mathcal{Y}_2$,*

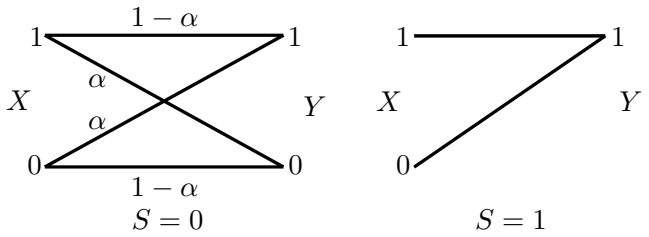

Figure 5.3: Example 5.2: A binary WOM with writing noise.

denoted as $W_1 \preceq W_2$*, if there exists a DMC* $W : \mathcal{Y}_2 \to \mathcal{Y}_1$ *such that* W *satisfies the equation* $W_1(y_1|x) = \sum_{y_2 \in \mathcal{Y}_2} W_2(y_2|x)W(y_1|y_2)$.

Next, we bring the required property of channels whose capacity is achieved by the limited construction to be proposed.

Property 5.1: *There exist functions* $p(v|s)$ *and* $x(v, s)$ *that maximize the Gelfand-Pinsker capacity in Theorem* 5.4 *and satisfy the condition* $p(y|v) \succeq p(s|v)$.

It is an open problem whether or not the model of Example 5.1 satisfies the degradation condition of Property 5.1. However, we can modify the model such that it will satisfy Property 5.1. Specifically, we study the following model:

Example 5.2: *Let the sets* $\mathcal{X}, \mathcal{S}$ *and* $\mathcal{Y}$ *be all equal to* $\{0, 1\}$*. The channel and state pmfs are given by* $p_S(1) = \beta$ *and*

$$p_{Y|XS}(1|x, s) = \begin{cases} \alpha & \text{if } (x, s) = (0, 0) \\ 1 - \alpha & \text{if } (x, s) = (1, 0) \\ 1 & \text{if } s = 1. \end{cases} \tag{5.9}$$

In words, if $s = 1$ *the channel output is always 1, and if* $s = 0$*, the channel behave as a binary symmetric channel. The cost function is given by* $b(x_i) = x_i$*. The error model is also presented in Figure* 5.3*. This model can represent a writing noise, as a cell of state* $s = 1$ *is not written on and it never suffers errors.*

We claim that the model of Example 5.2 satisfies the degradation condition of Property 5.1. To show this, we need first to find the functions $p(v|s)$ and $x(v,s)$ that maximize the Gelfand-Pinsker capacity in Theorem 5.4. Those functions are established in the following theorem.

Theorem 5.6: *The capacity of the channel in Example* 5.2 *is*

$$C = (1-\beta)[h(\epsilon * \alpha) - h(\alpha)],$$

where $\epsilon = B/(1-\beta)$ *and* $\epsilon * \alpha \equiv \epsilon(1-\alpha)+(1-\epsilon)\alpha$. *The selections* $\mathcal{V} = \{0,1\}$, $x(v,s) = v \wedge \neg s$ *(where* $\wedge$ *is the logical AND operation, and* $\neg$ *is the logical negation), and*

$$p_{V|S}(1|0) = \epsilon, \qquad p_{V|S}(1|1) = \frac{\epsilon(1-\alpha)}{\epsilon * \alpha} \tag{5.10}$$

achieve this capacity.

Theorem 5.6 is similar to [36, Theorem 4], and its proof is described in Section 5.5. Intuitively, the upper bound is obtained by assuming that the state information is available also at the decoder, and the lower bound is obtained by setting the functions $x(v,s)$ and $p(v|s)$ according to the statement of the theorem. The proof that the model in Example 5.2 satisfies the degradation condition of Property 5.1 is completed by the following lemma.

Lemma 5.1: *The capacity achieving functions of Theorem* 5.6 *for the model of Example* 5.2 *satisfy the degradation condition of Property* 5.1. *That is, the channel* $p(s|v)$ *is degraded with respect to the channel* $p(y|v)$.

Lemma 5.1 is proven in Section 5.5, and consequently the capacity of the model in Example 5.2 can be achieved by our limited construction. In the next subsection we describe the construction for channel models that satisfy Property 5.1, including the model in Example 5.2.

5.3.2 Multicoding Construction for Degraded Channels

Notice first that the capacity-achieving distribution of the asymmetric channel in Section 5.2 actually satisfies Property 5.1. In the asymmetric channel-coding case, the state can be thought of as a degenerate random variable (a RV which only takes a single value), and therefore we can choose W in Definition 5.1 to be degenerate as well, and by that satisfy Property 5.1. We will see that the construction that we present in this subsection is a generalization of Construction 5.1.

The construction has a similar structure as the achievability proof of the Gelfand-Pinsker Theorem (Theorem 5.4). The encoder first finds a vector $v_{[n]}$ in a similar manner to Construction 5.1, where the RV $X|Y$ in Construction 5.1 is replaced with $V|Y$, and the RV X is replaced with $V|S$. The vector $U_{[n]}$ is now the polarization of the vector $V_{[n]}$, meaning that $U_{[n]} = V_{[n]}G_n$. The RV V is taken according to the pmfs $p(v|s)$ that maximize the rate expression in Equation (5.7). The selection of the vector $u_{[n]}$ is illustrated in Figure 5.4. After the vector $u_{[n]}$ is chosen, each bit $i \in [n]$ in the codeword $x_{[n]}$ is calculated by the function $x_i(v_i, s_i)$ that maximizes Equation (5.7). To use the model of Example 5.2, one should use the functions $p(v|s)$ and $x(v, s)$ according to Theorem 5.6. The key to showing that the scheme achieves the channel capacity is that the fraction $|\mathcal{H}^c_{V|S} \cap \mathcal{L}^c_{V|Y}|/n$ can be shown to vanish for large n if the channel satisfies Property 5.1. Then, by the same intuition as in Equation (5.2) and using Equation (5.6), the replacements imply that the asymptotic rate of the codes is

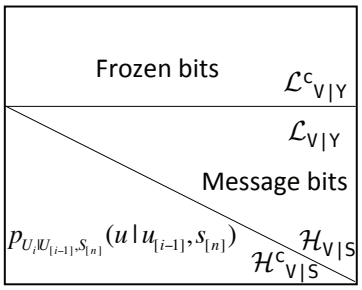

Figure 5.4: Encoding the vector $u_{[n]}$ in Construction 5.2.

$$\begin{aligned}|\mathcal{H}_{V|S} \cap \mathcal{L}_{V|Y}|/n &= 1 - |\mathcal{H}^c_{V|S}|/n - |\mathcal{L}^c_{V|Y}|/n + |\mathcal{H}^c_{V|S} \cap \mathcal{L}^c_{V|Y}|/n \\ &\to 1 - (1 - H(V|S)) - H(V|Y) + 0 \\ &= I(V;Y) - I(V;S),\end{aligned}$$

achieving the Gelfand-Pinsker capacity of Theorem 5.4. We now describe the coding scheme formally.

Construction 5.2:

Encoding

Input*: a message* $m_{[|\mathcal{H}_{V|S} \cap \mathcal{L}_{V|Y}|]} \in \{0,1\}^{|\mathcal{H}_{V|S} \cap \mathcal{L}_{V|Y}|}$ *and a state* $s_{[n]} \in \{0,1\}^n$.

Output: *a codeword* $x_{[n]} \in \{0,1\}^n$.

1. *For each* i *from* 1 *to* n*, assign*

$$u_i = \begin{cases} u \in \{0,1\} \quad \text{with probability } p_{U_i|U_{[i-1]},S_{[n]}}(u|u_{[i-1]}, s_{[n]}) & \text{if } i \in \mathcal{H}^c_{V|S} \\ m_{r(i,\mathcal{H}_{V|S} \cap \mathcal{L}_{V|Y})} & \text{if } i \in \mathcal{H}_{V|S} \cap \mathcal{L}_{V|Y} \\ f_{r(i,\mathcal{H}_{V|S} \cap \mathcal{L}^c_{V|Y})} & \text{if } i \in \mathcal{H}_{V|S} \cap \mathcal{L}^c_{V|Y}. \end{cases} \tag{5.11}$$

2. *Calculate* $v_{[n]} = u_{[n]} G_n$ *and for each* $i \in [n]$*, store the value* $x_i(v_i, s_i)$.

3. *Store the vector* $u_{\mathcal{H}^c_{V|S} \cap \mathcal{L}^c_{V|Y}}$ *separately using a point-to-point linear non-capacity-achieving polar code with a uniform input distribution. The encoder here does not use the state information in the encoding process, but rather treats it as an unknown part of the channel noise.*

Decoding

Input*: a noisy vector* $y_{[n]} \in \{0,1\}^n$.

Output: *a message estimation* $\hat{m}_{[|\mathcal{H}_{V|S} \cap \mathcal{L}_{V|Y}|]} \in \{0,1\}^{|\mathcal{H}_{V|S} \cap \mathcal{L}_{V|Y}|}$.

1. *Estimate the vector* $u_{\mathcal{H}^c_{V|S} \cap \mathcal{L}^c_{V|Y}}$ *by* $\hat{u}_{\mathcal{H}^c_{V|S} \cap \mathcal{L}^c_{V|Y}}$.

2. Estimate $u_{[n]}$ *by* $\hat{u}_{[n]}(y_{[n]}, f_{[|\mathcal{H}_{V|S}\cap\mathcal{L}^c_{V|Y}|]})$ *as follows: For each* i *from 1 to* n*, assign*

$$\hat{u}_i = \begin{cases} \arg\max_{u\in\{0,1\}} p_{U_i|U_{[i-1]},Y_{[n]}}(u|u_{[i-1]}, y_{[n]}) & \text{if } i \in \mathcal{L}_{V|Y} \\ \hat{u}_{r(i,\mathcal{H}^c_{V|S}\cap\mathcal{L}^c_{V|Y})} & \text{if } i \in \mathcal{H}^c_{V|S} \cap \mathcal{L}^c_{V|Y} \\ f_{r(i,\mathcal{H}_{V|S}\cap\mathcal{L}^c_{V|Y})} & \text{if } i \in \mathcal{H}_{V|S} \cap \mathcal{L}^c_{V|Y}. \end{cases} \tag{5.12}$$

3. Return the estimated message $\hat{m}_{[|\mathcal{H}_{V|S}\cap\mathcal{L}_{V|Y}|]} = \hat{u}_{\mathcal{H}_{V|S}\cap\mathcal{L}_{V|Y}}$.

The asymptotic performance of Construction 5.2 is stated in the following theorem.

Theorem 5.7: *If Property* 5.1 *holds, then Construction* 5.2 *achieves the capacity of Theorem* 5.4 *with a computational complexity of* $O(n \log n)$ *and a probability of decoding error of* $2^{-n^{1/2-\delta}}$ *for any* $\delta > 0$ *and large enough* n*.*

The proof of Theorem 5.7 is shown in Section 5.6. The next subsection describes a method of removing the degradation requirement of Property 5.1. This also allows one to achieve also the capacity of the more realistic model of Example 5.1.

5.3.3 Multicoding Construction without Degradation

A technique called "chaining" was proposed in [63] that allows one to achieve the capacity of models that do not exhibit the degradation condition of Property 5.1. The chaining idea was presented in the context of broadcast communication and point-to-point universal coding. We connect it here to the application of flash memory rewriting through Example 5.1. We note also that the chaining technique that follows comes with a price of a slower convergence to the channel capacity, and thus a lower non-asymptotic code rate.

The requirement of Construction 5.2 for degraded channels comes from the fact that the set $\mathcal{H}^c_{V|S} \cap \mathcal{L}^c_{V|Y}$ needs to be communicated to the decoder in a side channel. If the fraction $(1/n)|\mathcal{H}^c_{V|S} \cap \mathcal{L}^c_{V|Y}|$ vanishes with n, Construction 5.2 achieves the channel capacity. In this subsection we deal with the case that the fraction $(1/n)|\mathcal{H}^c_{V|S} \cap \mathcal{L}^c_{V|Y}|$ does not vanish. In

this case we have

$$\begin{aligned}|H_{V|S} \cap L_{V|Y}|/n =& 1 - |H^c_{V|S} \cup L^c_{V|Y}|/n \\ =& 1 - |H^c_{V|S}|/n - |L^c_{V|Y}|/n + |H^c_{V|S} \cap L^c_{V|Y}|/n \\ \to& I(V;Y) - I(V;S) + |H^c_{V|S} \cap L^c_{V|Y}|/n.\end{aligned}$$

The idea is then to store the subvector $u_{\mathcal{H}^c_{V|S} \cap \mathcal{L}^c_{V|Y}}$ in a subset of the indices $\mathcal{H}_{V|S} \cap \mathcal{L}_{V|Y}$ of an *additional code block* of n cells. The additional block uses the same coding technique as the original block. Therefore, it can use about $I(V;Y) - I(V;S)$ of the cells to store additional message bits, and by that approach the channel capacity. We denote by $\mathcal{R}$ the subset of $\mathcal{H}_{V|S} \cap \mathcal{L}_{V|Y}$ in which we store the the subvector $u_{\mathcal{H}^c_{V|S} \cap \mathcal{L}^c_{V|Y}}$ of the previous block. Note that the additional block also faces the same difficulty as the original block with the set $H^c_{V|S} \cap L^c_{V|Y}$. To solve this, we use the same solution, recursively, sending a total of k blocks, each of length n. Each block can store a source message of a fraction that approaches the channel capacity. The "problematic" bits of block k (the last block) will then be stored using yet another block, but this block will be coded without taking the state information into account, and thus will not face the same difficulty. The last block is thus causing a rate loss, but this loss is of a fraction $1/k$, which vanishes for large k. The decoding is performed "backwards", starting from the last block and ending with the first block. The chaining construction is illustrated in Figure 5.5. In the following formal description of the construction we denote the index i of the j-th block of the message by $m_{i,j}$, and similarly for

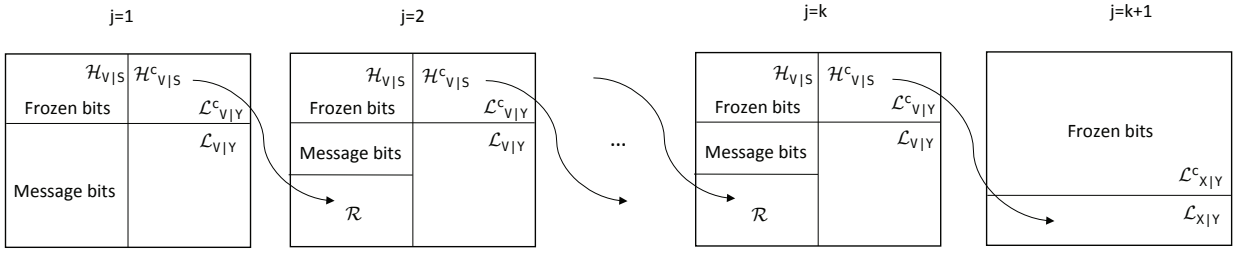

Figure 5.5: The chaining construction

other vectors. The vectors themselves are are also denoted in two dimensions, for example $x_{[n],[k]}$.

Construction 5.3: *Let $\mathcal{R}$ be an arbitrary subset of $\mathcal{H}_{V|S} \cap \mathcal{L}_{V|Y}$ of size $|\mathcal{H}^c_{V|S} \cap \mathcal{L}^c_{V|Y}|$.*

Encoding

***Input**: a message $m_{[(|\mathcal{H}_{V|S}\cap\mathcal{L}_{V|Y}|-|\mathcal{H}^c_{V|S}\cap\mathcal{L}^c_{V|Y}|],[k])} \in \{0,1\}^{k(|\mathcal{H}_{V|S}\cap\mathcal{L}_{V|Y}|-|\mathcal{H}^c_{V|S}\cap\mathcal{L}^c_{V|Y}|)}$ and a state $s_{[n],[k]} \in \{0,1\}^{kn}$.*

***Output:** a codeword $x_{[n],[k]} \in \{0,1\}^{kn}$.*

1. *Let $u_{[n],0} \in \{0,1\}^n$ be an arbitrary vector. For each j from 1 to k, and for each i from 1 to n, assign*

$$u_{i,j} = \begin{cases} u \in \{0,1\} \quad w.\ p.\ p_{U_i|U_{[i-1]},S_{[n]}}(u|u_{[i-1]}, s_{[n]}) & \text{if } i \in \mathcal{H}^c_{V|S} \\ m_{r(i,\mathcal{H}_{V|S}\cap\mathcal{L}_{V|Y}),j} & \text{if } i \in (\mathcal{H}_{V|S} \cap \mathcal{L}_{V|Y}) \setminus \mathcal{R} \\ u_{r(i,\mathcal{H}^c_{V|S}\cap\mathcal{L}^c_{V|Y}),j-1} & \text{if } i \in \mathcal{R} \\ f_{r(i,\mathcal{H}_{V|S}\cap\mathcal{L}^c_{V|Y}),j} & \text{if } i \in \mathcal{H}_{V|S} \cap \mathcal{L}^c_{V|Y}. \end{cases} \tag{5.13}$$

2. *For each j from 1 to k calculate $v_{[n],j} = u_{[n],j}G_n$, and for each $i \in [n]$, store the value $x_{i,j}(v_{i,j}, s_{i,j})$.*

3. *Store the vector $u_{\mathcal{H}^c_V\cap\mathcal{L}^c_{V|Y},k}$ separately using a point-to-point linear non-capacity-achieving polar code with a uniform input distribution. The encoder here does not use the state information in the encoding process, but rather treat it as an unknown part of the channel noise.*

Decoding

***Input**: a noisy vector $y_{[n],[k]} \in \{0,1\}^k\, n$.*

***Output:** a message estimation $\hat{m}_{[|\mathcal{H}_{V|S}\cap\mathcal{L}_{V|Y}|-|\mathcal{H}^c_{V|S}\cap\mathcal{L}^c_{V|Y}|],[k]} \in \{0,1\}^{k|\mathcal{H}_{V|S}\cap\mathcal{L}_{V|Y}|-|\mathcal{H}^c_{V|S}\cap\mathcal{L}^c_{V|Y}|}$.*

1. *Estimate the vector $u_{\mathcal{H}^c_{V|S}\cap\mathcal{L}^c_{V|Y},k}$ by $\hat{u}_{\mathcal{H}^c_{V|S}\cap\mathcal{L}^c_{V|Y},k}$, and let $\hat{u}_{\mathcal{R},k+1} = \hat{u}_{\mathcal{H}^c_{V|S}\cap\mathcal{L}^c_{V|Y},k}$.*

2. *Estimate* $u_{[n],[k]}$ *by* $\hat{u}_{[n],[k]}(y_{[n],[k]}, f_{[|\mathcal{L}^c_{V|Y} \cap \mathcal{H}_{V|S}|],[k]})$ *as follows: For each* j *down from* k *to* 1*, and for each* i *from 1 to* n*, assign*

$$\hat{u}_i^j = \begin{cases} \underset{u \in \{0,1\}}{\arg\max}\, p_{U_i|U_{[i-1]},Y_{[n]}}(u|u_{[i-1],j}, y_{[n],j}) & \text{if } i \in \mathcal{L}_{V|Y} \\ \hat{u}_{r(i,\mathcal{R}),j+1} & \text{if } i \in \mathcal{H}^c_{V|S} \cap \mathcal{L}^c_{V|Y} \\ f_{r(i,\mathcal{H}_{V|S} \cap \mathcal{L}^c_{V|Y}),j} & \text{if } i \in \mathcal{H}_{V|S} \cap \mathcal{L}^c_{V|Y}. \end{cases} \tag{5.14}$$

3. *Return the estimated message* $\hat{m}_{[|\mathcal{H}_{V|S} \cap \mathcal{L}_{V|Y}| - |\mathcal{H}^c_{V|S} \cap \mathcal{L}^c_{V|Y}|],[k]} = \hat{u}_{(\mathcal{H}_{V|S} \cap \mathcal{L}_{V|Y}) \setminus \mathcal{R},[k]}$.

Constructions 5.2 and 5.3 can also be used for communication over broadcast channels in Marton's region, as described in [29, 63]. Constructions 5.2 and 5.3 improve on these previous results since they provably achieve the capacity with linear storage requirement.

Construction 5.3 achieves the capacity of Theorem 5.4 with low complexity, without the degradation requirement of Property 5.1. This result was stated in Theorem 5.5. The proof of Theorem 5.5 follows from Theorem 5.7 and the fact that the rate loss vanishes with large k. Construction 5.3 is useful for the realistic model of flash memory-rewriting of Example 5.1, using the appropriate capacity-achieving functions $p(v|s)$ and $x(v, s)$.

5.4 Summary

In this chapter we proposed three capacity-achieving polar coding schemes, for the settings of asymmetric channel coding and flash memory rewriting. The scheme for asymmetric channels improves on the scheme of [37] by reducing the exponential storage requirement into a linear one. The idea for this reduction is to perform the encoding randomly instead of using Boolean functions, and to transmit a vanishing fraction of information on a side channel.

The second proposed scheme is used for the setting of flash memory rewriting. We propose a model of flash memory rewriting with writing noise, and show that the scheme achieves its capacity. We also describe a more general class of channels whose capacity can

be achieved using the scheme. The second scheme is derived from the asymmetric-channel scheme by replacing the Shannon random variables X and $X|Y$ with the Gelfand-Pinsker random variables $V|S$ and $V|Y$, respectively.

The last proposed scheme achieves the capacity of any channel with non-causal state information at the encoder. We bring a model of noisy flash memory rewriting for which the scheme would be useful. The main idea in this scheme is called code chaining. Side-information coding can be used in flash and phase-change memory (PCM) also in ways other than rewriting. Several such approaches were proposed recently based on the model of memory with defective cells [48, 49]. Therefore, the schemes proposed in this section can be useful also in the mentioned approaches. We note that the sparse-graph WOM schemes of Chapter 4 might also be used for these settings, with some modifications.

5.5 Capacity Proofs

5.5.1 Proof of Theorem 5.6

In this section we prove Theorem 5.6. A similar result was proven in [36, Theorem 4]. We show a different proof here, which we find more intuitive. Theorem 5.6 states that the capacity of the channel in Example 5.2 is

$$C = (1-\beta)[h(\epsilon * \alpha) - h(\alpha)],$$

where $\epsilon = B/(1-\beta)$ and $\epsilon * \alpha \equiv \epsilon(1-\alpha) + (1-\epsilon)\alpha$. An upper bound on the capacity can be obtained by assuming that the state information is available also to the decoder. In this case, the best coding scheme would ignore the cells with $s_i = 1$ (about a fraction β of the cells), and the rest of the cells would be coded according to a binary symmetric channel with an input cost constraint. It is optimal to assign a channel input $x_i = 0$ for the cells with state $s_i = 1$, such that those cells who do not convey information do not contribute to the cost. We now focus on the capacity of the binary symmetric channel with cost constraint. To comply

with the expected input cost constraint B of the channel of Example 5.2, the expected cost of the input to the binary symmetric channel (BSC) must be at most $\epsilon = B/(1-\beta)$. To complete the proof of the upper bound, we show next that the capacity of the BSC with cost constraint is equals to $h(\alpha * \epsilon) - h(\alpha)$. For this channel, we have

$$H(Y|X) = h(\alpha)p_X(0) + h(\alpha)p_X(1) = h(\alpha).$$

We are left now with maximizing the entropy $H(Y)$ over the input pmfs $p_X(1) \leq \epsilon$. We have

$$\begin{aligned} p_Y(1) =& p_{Y|X}(1|0)p_X(0) + p_{Y|X}(1|1)p_X(1) \\ =& \alpha(1 - p_X(1)) + (1-\alpha)p_X(1) \\ =& \alpha * p_X(1). \end{aligned}$$

Now since $p_X(1) \leq \epsilon \leq 1/2$ and $\alpha * p_X(1)$ is increasing in $p_X(1)$ below $1/2$, it follows that $p_Y(1) \leq \alpha * \epsilon \leq 1/2$ and therefore also that $H(Y) \leq h(\alpha * \epsilon)$. So we have

$$\max_{p_X(1)\leq\epsilon} I(X;Y) = \max_{p_X(1)\leq\epsilon} (H(Y) - H(Y|X)) = h(\alpha * \epsilon) - h(\alpha).$$

This completes the proof of the upper bound.

The lower bound is obtained by considering the selections $\mathcal{V} = \{0,1\}$, $x(v,0) = v$, $x(v,1) = 0$, and

$$p_{V|S}(1|0) = \epsilon, \qquad p_{V|S}(1|1) = \frac{\epsilon(1-\alpha)}{\epsilon * \alpha}, \tag{5.15}$$

and calculating the rate expression directly. Notice first that the cost constraint is met since

$$p_X(1) = p_{X|S}(1|0)p_S(0) = p_{V|S}(1|0)p_S(0) = \epsilon(1-\beta) = B.$$

We need to show that $H(V|S) - H(V|Y) = (1-\beta)[H(\alpha * \epsilon) - H(\alpha)]$. Given the distributions

p_S and $p_{V|S}$, the conditional entropy $H(V|S)$ is

$$\begin{aligned} H(V|S) = & \sum_{s\in\{0,1\}} p_S(s)H(V|S=s) \\ =& p_S(0)H(V|S=0) + p_S(1)H(V|S=1) \\ =& (1-\beta)H(\epsilon) + \beta H\left(\frac{\epsilon(1-\alpha)}{\epsilon * \alpha}\right) \end{aligned}$$

To compute the conditional entropy $H(V|Y)$, we first compute the probability distribution of the memory output Y as follows:

$$\begin{aligned} p_Y(0) = & \sum_{v\in\{0,1\}} p_{Y|VS}(0|v,0)p_{V|S}(v|0)p_S(0) \\ =& (1-\beta)((1-\alpha)(1-\epsilon) + \alpha\epsilon) \\ =& (1-\beta)(\alpha * (1-\epsilon)), \\ p_Y(1) =& 1 - p_Y(0) \\ =& (1-\beta)(\alpha * \epsilon) + \beta. \end{aligned}$$

The conditional distribution $p_{V|Y}$ is given by

$$\begin{aligned}
p_{V|Y}(1|0) &= \sum_{s\in\{0,1\}} p_{VS|Y}(1,s|0) \\
&= \sum_{s\in\{0,1\}} \frac{p_{Y|VS}(0|1,s)p_{VS}(1,s)}{p_Y(0)} \\
&= \sum_{s\in\{0,1\}} \frac{p_{Y|VS}(0|1,s)p_{V|S}(1|s)p_S(s)}{p_Y(0)} \\
&= \frac{\alpha\epsilon}{\alpha * (1-\epsilon)}, \\
p_{V|Y}(1|1) &= \sum_{s\in\{0,1\}} p_{VS|Y}(1,s|1) \\
&= \sum_{s\in\{0,1\}} \frac{p_{Y|VS}(1|1,s)p_{VS}(1,s)}{p_Y(1)} \\
&= \sum_{s\in\{0,1\}} \frac{p_{Y|VS}(1|1,s)p_{V|S}(1|s)p_S(s)}{p_Y(1)} \\
&= \frac{(1-\alpha)\epsilon(1-\beta) + \frac{\epsilon(1-\alpha)}{\epsilon*\alpha}\beta}{(1-\beta)(\alpha*\epsilon)+\beta} \\
&= \frac{\epsilon(1-\alpha)}{\epsilon*\alpha}.
\end{aligned}$$

Therefore we have

$$\begin{aligned}
H(V|Y) &= \sum_{y\in\{0,1\}} p_Y(y)H(V|Y=y) \\
&= (1-\beta)(\alpha*(1-\epsilon))H\left(\frac{\alpha\epsilon}{\alpha*(1-\epsilon)}\right) + (\beta+(1-\beta)(\alpha*\epsilon))H\left(\frac{\epsilon(1-\alpha)}{\epsilon*\alpha}\right),
\end{aligned}$$

and then

$$\begin{aligned}
H(V|S) - H(V|Y) =&(1-\beta)\left[H(\epsilon) - (\alpha * (1-\epsilon))H\left(\frac{\alpha\epsilon}{\alpha * (1-\epsilon)}\right) - (\alpha * \epsilon)H\left(\frac{\epsilon(1-\alpha)}{\epsilon * \alpha}\right)\right] \\
=&(1-\beta)\Bigg[H(\epsilon) + \alpha\epsilon\log_2\frac{\alpha\epsilon}{\alpha * (1-\epsilon)} + (1-\alpha)(1-\epsilon)\log_2\frac{(1-\alpha)(1-\epsilon)}{\alpha * (1-\epsilon)} \\
&+ \alpha(1-\epsilon)\log_2\frac{\alpha(1-\epsilon)}{\alpha * \epsilon} + \epsilon(1-\alpha)\log_2\frac{\epsilon(1-\alpha)}{\alpha * \epsilon}\Bigg] \\
=&(1-\beta)[H(\alpha * \epsilon) + H(\epsilon) + \alpha\epsilon\log_2(\alpha\epsilon) + (1-\alpha)(1-\epsilon)\log_2(1-\alpha)(1-\epsilon) \\
&+ \alpha(1-\epsilon)\log_2\alpha(1-\epsilon) + \epsilon(1-\alpha)\log_2\epsilon(1-\alpha)] \\
=&(1-\beta)\left[H(\alpha * \epsilon) + H(\epsilon) - H(\alpha) - H(\epsilon)\right] \\
=&(1-\beta)\left[H(\alpha * \epsilon) - H(\alpha)\right].
\end{aligned}$$

5.5.2 Proof of Lemma 5.1

In this subsection we prove Lemma 5.1. We need to show that, using the functions of Therem 5.6, there exists a DMC $W : \{0,1\}^2 \to \{0,1\}^2$ such that

$$p_{S|V}(s|v) = \sum_{y\in\{0,1\}^2} p_{Y|V}(y|v)W(s|y). \tag{5.16}$$

To define such channel W, we first claim that

$$p_{Y|V,S}(1|v,0)p_{V|S}(x|0) = (\epsilon * \alpha)p_{V|S}(x|1). \tag{5.17}$$

Equation (5.17) follows directly from Equation (5.10) since

$$\frac{p_{Y|V,S}(1|0,0)p_{V|S}(0|0)}{p_{V|S}(0|1)} = \frac{\alpha(1-\epsilon)}{\frac{\alpha(1-\epsilon)}{\epsilon*\alpha}} = \epsilon * \alpha,$$

$$\frac{p_{Y|V,S}(1|1,0)p_{V|S}(1|0)}{p_{V|S}(1|1)} = \frac{(1-\alpha)\epsilon}{\frac{(1-\alpha)\epsilon}{\epsilon*\alpha}} = \epsilon * \alpha.$$

Next, we claim that $\frac{p_{V,S}(v,1)}{p_{V,Y}(v,1)} = \frac{\beta}{(\epsilon * \alpha)(1-\beta)+\beta}$ for any $v \in \{0,1\}$, and therefore that $\frac{p_{V,S}(v,1)}{p_{V,Y}(v,1)} \in [0,1]$. This follows from

$$\begin{aligned}
\frac{p_{V,S}(v,1)}{p_{V,Y}(v,1)} &\overset{(a)}{=} \frac{p_{V|S}(v|1)p_S(1)}{p_{Y,V|S}(1,v|0)p_S(0) + p_{Y,V|S}(1,v|1)p_S(1)} \\
&\overset{(b)}{=} \frac{p_{V|S}(v|1)\beta}{p_{Y|V,S}(1|v,0)p_{V|S}(v|0)(1-\beta) + p_{Y|V,S}(1|v,1)p_{V|S}(x|1)\beta} \\
&\overset{(c)}{=} \frac{p_{V|S}(v|1)\beta}{(\epsilon * \alpha)p_{V|S}(v|1)(1-\beta) + p_{V|S}(v|1)\beta} \\
&= \frac{\beta}{(\epsilon * \alpha)(1-\beta)+\beta},
\end{aligned}$$

where (a) follows from the law of total probability, (b) follows from the definition of conditional probability, and (c) follows from Equations (5.9) and (5.17).

Since $\frac{p_{V,S}(v,1)}{p_{V,Y}(v,1)}$ is not a function of x and is in $[0,1]$, we can define W as following:

$$W(s|y) \triangleq \begin{cases} 1 & \text{if } (s,y) = (0,0) \\ 1 - \frac{p_{S|V}(1|v)}{p_{Y|V}(1|v)} & \text{if } (s,y) = (0,1) \\ \frac{p_{S|V}(1|v)}{p_{Y|V}(1|v)} & \text{if } (s,y) = (1,1) \\ 0 & \text{if } (s,y) = (1,0). \end{cases}$$

We show next that Equation (5.16) holds for W defined above:

$$\begin{aligned}
\sum_{y \in \{0,1\}} p_{Y|V}(y|v)W(s|y) =& p_{Y|V}(0|v)W(s|0) + p_{Y|V}(1|v)W(s|1) \\
=& \left[p_{Y|V}(0|v) + p_{Y|V}(1|v)\left(1 - \frac{p_{S|V}(1|v)}{p_{Y|V}(1|v)}\right)\right] \mathbb{1}[s=0] + p_{S|V}(1|v)\mathbb{1}[s=1] \\
=& \left[1 - p_{S|V}(1|v)\right]\mathbb{1}[s=0] + p_{S|V}(1|v)\mathbb{1}[s=1] \\
=& p_{S|V}(0|v)\mathbb{1}[s=0] + p_{S|V}(1|v)\mathbb{1}[s=1] \\
=& p_{S|V}(s|v).
\end{aligned}$$

So the channel W satisfies Equation (5.16) and thus the lemma holds.

5.6 Proof of Theorem 5.7

In this section we prove Theorem 5.7. The complexity claim of Theorem 5.7 is explained in [37, Section III.B]. We start with the asymptotic rate of Construction 5.2. We want to show that $\lim_{n\to\infty}(1/n)|\mathcal{H}^c_{V|S} \cap \mathcal{L}^c_{V|Y}| = 0$. Since $p_{S|V}$ is degraded with respect to $p_{Y|V}$, it follows from [29, Lemma 4] that $\mathcal{L}_{V|S} \subseteq \mathcal{L}_{V|Y}$, and therefore that $\mathcal{L}^c_{V|S} \supseteq \mathcal{L}^c_{V|Y}$. So we have

$$\lim_{n\to\infty}(1/n)|\mathcal{H}^c_{V|S} \cap \mathcal{L}^c_{V|Y}| \leq \lim_{n\to\infty}(1/n)|\mathcal{H}^c_{V|S} \cap \mathcal{L}^c_{V|S}| = 0,$$

where the equality follows from the definition of the sets. To complete the proof of the theorem, we need to show that for some frozen vector, the input cost meets the design constraint, while the decoding error probability vanishes sub-exponentially fast in the block length. We show this result using the probabilistic method. That is, we first assume that the frozen vector is random and drawn uniformly, and analyze the input cost and error probability in this case, and then we show the existence of a good vector using Markov's inequalities and the union bound. Denote the uniformly distributed random vector that represents the frozen vector by $F_{[|\mathcal{H}_{V|S}\cap\mathcal{L}^c_{V|Y}|]}$. We show next that the expected input cost exceeds the design constraint B by a vanishing amount.

5.6.1 Expected Input Cost

Define

$$b^n(X_{[n]}) = \sum_{i=1}^{n} b(X_i).$$

For a state vector $s_{[n]}$ and the encoding rule (5.11) with uniform frozen vector $F_{[|\mathcal{H}_{V|S}\cap\mathcal{L}^c_{V|Y}|]}$, each vector $u_{[n]}$ appears with probability

$$\left(\prod_{i\in\mathcal{H}^c_{V|S}} p_{U_i|U_{[i-1]},S_{[n]}}(u_i|u_{[i-1]}, s_{[n]})\right) 2^{-|\mathcal{H}_{V|S}|}.$$

Remember that the joint pmf $p_{S_{[n]},U_{[n]}}$ refers to the capacity-achieving distribution of the channel. The expected cost is expressed as

$$\mathsf{E}_{F_{[|\mathcal{H}_{V|S}\cap\mathcal{L}^c_{V|Y}|]}}(b^n(X_{[n]})) = \sum_{u_{[n]},s_{[n]}} p_{S_{[n]}}(s_{[n]}) \left(\prod_{i\in\mathcal{H}^c_{V|S}} p_{U_i|U_{[i-1]},S_{[n]}}(u_i|u_{[i-1]},s_{[n]}) \right) 2^{-|\mathcal{H}_{V|S}|} b^n(u_{[n]}G_n).$$

Define the joint pmf:

$$q_{S_{[n]},U_{[n]}} \equiv p_{S_{[n]}}(s_{[n]}) \left(\prod_{i\in\mathcal{H}^c_{V|S}} p_{U_i|U_{[i-1]},S_{[n]}}(u_i|u_{[i-1]},s_{[n]}) \right) 2^{-|\mathcal{H}_{V|S}|}. \tag{5.18}$$

Intuitively, $q_{S_{[n]},U_{[n]}}$ refers to the distribution imposed by the encoding procedure of Construction 5.2. Then we have

$$\begin{aligned}
\mathsf{E}_{F_{[|\mathcal{H}_{V|S}\cap\mathcal{L}^c_{V|Y}|]}}(b^n(X_{[n]})) =& \mathsf{E}_{q_{S_{[n]},U_{[n]}}}[b^n(U_{[n]}G_n)] \\
\leq& \mathsf{E}_{p_{S_{[n]},U_{[n]}}}[b^n(U_{[n]}G_n)] + \max_x b(x)\|p_{S_{[n]},U_{[n]}} - q_{S_{[n]},U_{[n]}}\| \\
=& nB + \max_x b(x)\|p_{S_{[n]},U_{[n]}} - q_{S_{[n]},U_{[n]}}\|,
\end{aligned}$$

where $\|\cdot\|$ is the L_1 distance and the inequality follows from the triangle inequality. We will now prove that

$$\mathsf{E}_{F_{[|\mathcal{H}_{V|S}\cap\mathcal{L}^c_{V|Y}|]}}(b^n(X_{[n]})) \leq nB + 2^{-n^{1/2-\delta}},$$

by showing that $\|p_{U_{[n]},S_{[n]}} - q_{U_{[n]},S_{[n]}}\| \leq 2^{-n^{1/2-\delta}}$. To do this, we will prove a slightly stronger relation that will be used also for the proof of the probability of decoding error. We first define the joint pmf:

$$q_{S_{[n]},U_{[n]},Y_{[n]}} \equiv q_{U_{[n]},S_{[n]}}(u_{[n]},s_{[n]}) p_{Y_{[n]}|U_{[n]},S_{[n]}}(y_{[n]}|u_{[n]},s_{[n]}).$$

Then we notice that

$$\begin{aligned}\|p_{U_{[n]},S_{[n]}} - q_{U_{[n]},S_{[n]}}\| &= \sum_{u_{[n]},s_{[n]}} |p(u_{[n]},s_{[n]}) - q(u_{[n]},s_{[n]})| \\ &= \sum_{u_{[n]},s_{[n]}} \left| \sum_{y_{[n]}} [p(s_{[n]},u_{[n]},y_{[n]}) - q(s_{[n]},u_{[n]},y_{[n]})] \right| \\ &\leq \sum_{s_{[n]},u_{[n]},y_{[n]}} |p(s_{[n]},u_{[n]},y_{[n]}) - q(s_{[n]},u_{[n]},y_{[n]})| \\ &= \|p_{S_{[n]},U_{[n]},Y_{[n]}} - q_{S_{[n]},U_{[n]},Y_{[n]}}\|,\end{aligned}$$

where the inequality follows from the triangle inequality. The proof of the expected cost is completed with the following lemma, which will be used also for analyzing the probability of decoding error:

Lemma 5.2:

$$\|p_{S_{[n]},U_{[n]},Y_{[n]}} - q_{S_{[n]},U_{[n]},Y_{[n]}}\| \leq 2^{-n^{1/2-\delta}}. \tag{5.19}$$

PROOF: Let $D(\cdot\|\cdot)$ denote the relative entropy. Then

$$
\begin{aligned}
&\|p_{S_{[n]},U_{[n]},Y_{[n]}} - q_{S_{[n]},U_{[n]},Y_{[n]}}\| \\
&\quad = \sum_{s_{[n]},u_{[n]},y_{[n]}} |p(s_{[n]},u_{[n]},y_{[n]}) - q(s_{[n]},u_{[n]},y_{[n]})| \\
&\quad \overset{(a)}{=} \sum_{s_{[n]},u_{[n]},y_{[n]}} |p(u_{[n]}|s_{[n]}) - q(u_{[n]}|s_{[n]})| p(s_{[n]}) p(y_{[n]}|u_{[n]},s_{[n]}) \\
&\quad \overset{(b)}{=} \sum_{s_{[n]},u_{[n]},y_{[n]}} \left| \prod_{i=1}^{n} p(u_i|u_{[i-1]},s_{[n]}) - \prod_{i=1}^{n} q(u_i|u_{[i-1]},s_{[n]}) \right| p(s_{[n]}) p(y_{[n]}|u_{[n]},s_{[n]}) \\
&\quad \overset{(c)}{=} \sum_{s_{[n]},u_{[n]},y_{[n]}} \left| \sum_{i} [p(u_i|u_{[i-1]},s_{[n]}) - q(u_i|u_{[i-1]},s_{[n]})] \right| p(s_{[n]}) p(y_{[n]}|u_{[n]},s_{[n]}) \\
&\qquad \cdot \prod_{j=1}^{i-1} p(u_j|u_{[j-1]},s_{[n]}) \prod_{j=i+1}^{n} q(u_j|u_{[j-1]},s_{[n]}) \\
&\quad \overset{(d)}{\leq} \sum_{i \in \mathcal{H}_{V|S}} \sum_{s_{[n]},u_{[n]},y_{[n]}} \left| p(u_i|u_{[i-1]},s_{[n]}) - q(u_i|u_{[i-1]},s_{[n]}) \right| p(s_{[n]}) \prod_{j=1}^{i-1} p(u_j|u_{[j-1]},s_{[n]}) \\
&\qquad \cdot \prod_{j=i+1}^{n} q(u_j|u_{[j-1]},s_{[n]}) p(y_{[n]}|u_{[n]},s_{[n]}) \\
&\quad = \sum_{i \in \mathcal{H}_{V|S}} \sum_{s_{[n]},u_{[i]}} \left| p(u_i|u_{[i-1]},s_{[n]}) - q(u_i|u_{[i-1]},s_{[n]}) \right| \prod_{j=1}^{i-1} p(u_j|u_{[j-1]},s_{[n]}) p(s_{[n]}) \\
&\quad \overset{(e)}{=} \sum_{i \in \mathcal{H}_{V|S}} \sum_{s_{[n]},u_{[i-1]}} p(u_{[i-1]},s_{[n]}) \|p_{U_i|U_{[i-1]}=u_{[i-1]},S_{[n]}=s_{[n]}} - q_{U_i|U_{[i-1]}=u_{[i-1]},S_{[n]}=s_{[n]}}\| \\
&\quad \overset{(f)}{\leq} \sum_{i \in \mathcal{H}_{V|S}} \sum_{s_{[n]},u_{[i-1]}} p(u_{[i-1]},s_{[n]}) \sqrt{2 \ln 2} \sqrt{D(p_{U_i|U_{[i-1]}=u_{[i-1]},S_{[n]}=s_{[n]}} \| q_{U_i|U_{[i-1]}=u_{[i-1]},S_{[n]}=s_{[n]}})} \\
&\quad \overset{(g)}{\leq} \sum_{i \in \mathcal{H}_{V|S}} \sqrt{(2 \ln 2) \sum_{s_{[n]},u_{[i-1]}} p(u_{[i-1]},s_{[n]}) D(p_{U_i|U_{[i-1]}=u_{[i-1]},S_{[n]}=s_{[n]}} \| q_{U_i|U_{[i-1]}=u_{[i-1]},S_{[n]}=s_{[n]}})} \\
&\quad = \sum_{i \in \mathcal{H}_{V|S}} \sqrt{(2 \ln 2) D(p_{U_i} \| q_{U_i}|U_{[i-1]},S_{[n]})} \\
&\quad \overset{(h)}{=} \sum_{i \in \mathcal{H}_{V|S}} \sqrt{(2 \ln 2)[1 - H(U_i|U_{[i-1]},S_{[n]})]},
\end{aligned}
$$

where

1. follows from the fact that $p(s_{[n]}) = q(s_{[n]})$ and $p(y_{[n]}|u_{[n]}, s_{[n]}) = q(y_{[n]}|u_{[n]}, s_{[n]})$,

2. follows from the chain rule,

3. follows from the telescoping expansion

$$\begin{aligned} B_{[n]} - A_{[n]} &= \sum_{i=1}^{n} A_{[i-1]} B_{[i:n]} - \sum_{i=1}^{n} A_{[i]} B_{[i+1:n]} \\ &= \sum_{i=1}^{n} (B_i - A_i) A_{[i-1]} B_{[i+1:n]}, \end{aligned}$$

   where $A_{[j:k]}$ and $B_{[j:k]}$ denote the products $\prod_{i=j}^{k} A_i$ and $\prod_{i=j}^{k} B_i$, respectively,

4. follows from the triangular inequality and the fact that $p(u_i|u_{[i-1]}, s_{[n]}) = q(u_i|u_{[i-1]}, s_{[n]})$ for all $i \in \mathcal{H}^c_{V|S}$ (according to Equation (5.18)),

5. follows from the chain rule again,

6. follows from Pinsker's inequality (see, e.g., [13, Lemma 11.6.1]),

7. follows from Jensen's inequality and

8. follows from the facts that $q(u_i|u_{[i-1]}, s_{[n]}) = 1/2$ for $i \in \mathcal{H}_{V|S}$ and from [29, Lemma 10].

Now if $i \in \mathcal{H}_{V|S}$, we have

$$\begin{aligned} 1 - H(U_i|U_{[i-1]}, S_{[n]}) &\leq 1 - [Z(U_i|U_{[i-1]}, S_{[n]})]^2 \\ &\leq 2^{-2n^{1/2-\delta}}, \end{aligned} \tag{5.20}$$

where the first inequality follows from Proposition 5.1, and the second inequality follows from the fact that i is in $\mathcal{H}_{V|S}$. This completes the proof of the lemma. ∎

5.6.2 Probability of Decoding Error

Let $\mathcal{E}_i$ be the set of pairs of vectors $(u_{[n]}, y_{[n]})$ such that $\hat{u}_{[n]}$ is a result of decoding $y_{[n]}$, and $\hat{u}_{[i]}$ satisfies both $\hat{u}_{[i-1]} = u_{[i-1]}$ and $\hat{u}_i \neq u_i$. The block decoding error event is given by $\mathcal{E} \equiv \cup_{i \in \mathcal{L}_{V|Y}} \mathcal{E}_i$. Under decoding given in (5.12) with an arbitrary tie-breaking rule, every pair $(u_{[n]}, y_{[n]}) \in \mathcal{E}_i$ satisfies

$$p_{U_i|U_{[i-1]},Y_{[n]}}(u_i|u_{[i-1]}, y_{[n]}) \leq p_{U_i|U_{[i-1]},Y_{[n]}}(u_i \oplus 1|u_{[i-1]}, y_{[n]}). \tag{5.21}$$

Consider the block decoding error probability $p_e(F_{[|\mathcal{H}_{V|S} \cap \mathcal{L}^c_{V|Y}|]})$ for the random frozen vector $F_{[|\mathcal{H}_{V|S} \cap \mathcal{L}^c_{V|Y}|]}$. For a state vector $s_{[n]}$ and the encoding rule (5.11), each vector $u_{[n]}$ appears with probability

$$\left(\prod_{i \in \mathcal{H}^c_{V|S}} p_{U_i|U_{[i-1]},S_{[n]}}(u_i|u_{[i-1]}, s_{[n]}) \right) 2^{-|\mathcal{H}_{V|S}|}.$$

By the definition of conditional probability and the law of total probability, the probability of error $p_e(F_{[|\mathcal{H}_{V|S} \cap \mathcal{L}^c_{V|Y}|]})$ is given by

$$\begin{aligned} \mathsf{E}_{F_{[|\mathcal{H}_{V|S} \cap \mathcal{L}^c_{V|Y}|]}}[p_e] = \sum_{u_{[n]}, s_{[n]}, y_{[n]}} & p_{S_{[n]}}(s_{[n]}) \left(\prod_{i \in \mathcal{H}^c_{V|S}} p_{U_i|U_{[i-1]},S_{[n]}}(u_i|u_{[i-1]}, s_{[n]}) \right) 2^{-|\mathcal{H}_{V|S}|} \\ & \cdot p_{Y_{[n]}|U_{[n]},S_{[n]}}(y_{[n]}|u_{[n]}, s_{[n]}) \mathbb{1}[(u_{[n]}, y_{[n]}) \in \mathcal{E}]. \end{aligned}$$

Then we have

$$\begin{aligned} \mathsf{E}_{F_{[|\mathcal{H}_{V|S} \cap \mathcal{L}^c_{V|Y}|]}}[p_e] =& q_{U_{[n]},Y_{[n]}}(\mathcal{E}) \\ \leq& \| q_{U_{[n]},Y_{[n]}} - p_{U_{[n]},Y_{[n]}} \| + p_{U_{[n]},Y_{[n]}}(\mathcal{E}) \\ \leq& \| q_{U_{[n]},Y_{[n]}} - p_{U_{[n]},Y_{[n]}} \| + \sum_{i \in \mathcal{L}_{V|Y}} p_{U_{[n]},Y_{[n]}}(\mathcal{E}_i), \end{aligned}$$

where the first inequality follows from the triangle inequality. Each term in the summation is bounded by

$$\begin{aligned}
p_{U_{[n]},Y_{[n]}}(\mathcal{E}_i) &\leq \sum_{u_{[i]},y_{[n]}} p(u_{[i]},y_{[n]}) \mathbb{1}\left[p(u_i|u_{[i-1]},y_{[n]}) \leq p(u_i \oplus 1|u_{[i-1]},y_{[n]})\right] \\
&\leq \sum_{u_{[i]},y_{[n]}} p(u_{[i-1]},y_{[n]}) p(u_i|u_{[i-1]},y_{[n]}) \sqrt{\frac{p(u_i \oplus 1|u_{[i-1]},y_{[n]})}{p(u_i|u_{[i-1]},y_{[n]})}} \\
&= Z(U_i|U_{[i-1]},Y_{[n]}) \\
&\leq 2^{-n^{1/2-\delta}},
\end{aligned}$$

where the last inequality follows from the fact that i belongs to the set $\mathcal{L}_{V|Y}$.

To prove that $\mathsf{E}_{F_{[|\mathcal{H}_{V|S} \cap \mathcal{L}^c_{V|Y}|]}}[p_e] \leq 2^{-n^{1/2-\delta'}}$ for some $\delta' > \delta$, we are left with showing that $\|p_{U_{[n]},Y_{[n]}} - q_{U_{[n]},Y_{[n]}}\| \leq 2^{-n^{1/2-\delta}}$. Notice that

$$\begin{aligned}
2\|p_{U_{[n]},Y_{[n]}} - q_{U_{[n]},Y_{[n]}}\| &= \sum_{u_{[n]},y_{[n]}} |p(u_{[n]},y_{[n]}) - q(u_{[n]},y_{[n]})| \\
&= \sum_{u_{[n]},y_{[n]}} \left| \sum_{s_{[n]}} [p(s_{[n]},u_{[n]},y_{[n]}) - q(s_{[n]},u_{[n]},y_{[n]})] \right| \\
&\leq \sum_{s_{[n]},u_{[n]},y_{[n]}} |p(s_{[n]},u_{[n]},y_{[n]}) - q(s_{[n]},u_{[n]},y_{[n]})| \\
&= 2\|p_{S_{[n]},U_{[n]},Y_{[n]}} - q_{S_{[n]},U_{[n]},Y_{[n]}}\|,
\end{aligned}$$

where the inequality follows from the triangle inequality. Lemma 5.2 now completes the proof that $\mathsf{E}_{F_{[|\mathcal{H}_{V|S} \cap \mathcal{L}^c_{V|Y}|]}}[p_e] = 2^{-n^{1/2-\delta}}$.

5.6.3 Existence of a Good Frozen Vector

We showed that

$$\mathsf{E}_{F_{[|\mathcal{H}_{V|S} \cap \mathcal{L}^c_{V|Y}|]}}(b^n(X_{[n]})) \leq nB + 2^{-n^{1/2-\delta}}.$$

In this subsection we will be interested to find a frozen vector which satisfies a slightly higher expected cost, namely

$$\mathsf{E}(b^n(X_{[n]})) \leq nB + 1/n^2.$$

Remember that B is a constant. We take a uniformly distributed frozen vector, and bound the probability that the expected cost exceeds $nB + 1/n^2$, using Markov's inequality. The following inequalities hold for large enough n, and we are not concerned here with the exact values required:

$$\begin{aligned}
P(\mathsf{E}(b^n(X_{[n]})) \geq nB + 1/n^2) &\leq \mathsf{E}_{F_{[|\mathcal{H}_{V|S} \cap \mathcal{L}^c_{V|Y}|]}}(b^n(X))/(nB + 1/n^2) \\
&\leq (nB + 2^{-n^{1/2-\delta}})/(nB + 1/n^2) \\
&\leq (nB + n^{-3})/(nB + n^{-2}) \\
&= 1 - (n^{-2} - n^{-3})/(nB + n^{-2}) \\
&\leq 1 - (n^{-2.1})/(nB + n^{-2}) \\
&\leq 1 - 1/(Bn^{3.1} + n^{0.1}) \\
&\leq 1 - 1/(Bn^{3.2}) \\
&\leq 1 - 1/n^4.
\end{aligned}$$

We now apply Markov's inequality on the probability of decoding error:

$$P(p_e \geq n^5 2^{-n^{1/2-\delta}}) \leq \frac{E_{F_{[|\mathcal{H}_{V|S} \cap \mathcal{L}^c_{V|Y}|]}}(p_e)}{n^5 2^{-n^{1/2-\delta}}} \leq \frac{2^{-n^{1/2-\delta}}}{n^5 2^{-n^{1/2-\delta}}} = 1/n^5.$$

By the union bound, the probability that either $\mathsf{E}(b^n(X_{[n]})) \geq nB + 1/n^2$ or $p_e \geq n^5 2^{-n^{1/2-\delta}}$ is at most

$$1 - 1/n^4 + 1/n^5 \leq 1 - 1/n^5.$$

This implies that the probability that both $\mathsf{E}(b^n(X_{[n]})) \leq nB + 1/n^2$ and $p_e \leq n^5 2^{-n^{1/2-\delta}}$ is at least $1/n^5$. So there exists at least one frozen vector for which the desired properties for both the expected cost and the probability of error hold. Furthermore, such a vector

can be found by repeatedly selecting a frozen vector uniformly at random until the required properties hold. The properties can be verified efficiently with close approximations by the upgradation and degradation method proposed in [81]. The expected number of times until a good vector is found is polynomial in the block length (at most n^5). This completes the proof of Theorem 5.7.

Part IV

Local Rank Modulation

Chapter 6

Rewriting with Gray Codes

The material in this chapter was published in [19, 20].

6.1 Definitions and Notation

6.1.1 Local Rank Modulation

Assume we have a set of t flash memory cells which we number $0, 1, \ldots, t-1$. Let us consider a sequence of t real-valued variables, $\mathbf{c} = (c_0, c_1, \ldots, c_{t-1}) \in \mathbb{R}^t$, where c_i denotes the charge level measured in the ith flash memory cell. We further assume $c_i \neq c_j$ for all $i \neq j$. The t variables induce a permutation $f_{\mathbf{c}} \in S_t$, where S_t denotes the set of all permutations over $[t] = \{0, 1, 2, \ldots, t-1\}$. The permutation $f_{\mathbf{c}}$ is defined as

$$f_{\mathbf{c}}(i) = |\{j \mid c_j < c_i\}| .$$

Thus, $f_{\mathbf{c}}(i)$ is the rank of the ith cell in ascending order.

We now turn to consider a larger set of $n \geq t$ flash memory cells. Again, we have a sequence of n variables, $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1}) \in \mathbb{R}^n$ where c_i denotes the measured charge level in the ith flash memory cell. We define a window of size t at position p to be

$$\mathbf{c}_{p,t} = (c_p, c_{p+1}, \ldots, c_{p+t-1}),$$

where the indices are taken modulo n, and also $0 \leq p \leq n-1$, and $1 \leq t \leq n$. We now define the (s,t,n)-*local rank-modulation (LRM) scheme*, which we do by defining the *demodulation* process. Let $s \leq t \leq n$ be positive integers, with $s|n$. Given a sequence of n distinct real-valued variables, $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$, the demodulation maps $\mathbf{c}$ to the sequence of n/s permutations from S_t as follows:

$$\mathbf{f_c} = (f_{\mathbf{c}_{0,t}}, f_{\mathbf{c}_{s,t}}, f_{\mathbf{c}_{2s,t}}, \ldots, f_{\mathbf{c}_{n-s,t}}). \tag{6.1}$$

Loosely speaking, we scan the n variables using windows of size t positioned at multiples of s and write down the permutations from S_t induced by the *local* views of the sequence.

In the context of flash-memory storage devices, we shall consider the n variables, $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$, to be the charge-level readings from n flash cells. The demodulated sequence, $\mathbf{f_c}$, will stand for the original information which was stored in the n cells. This approach will serve as the main motivation for this chapter, as it was also for [19,45,46,82,85]. See Figure 6.1 for an example of a demodulation of a $(3,5,12)$-locally rank-modulated signal.

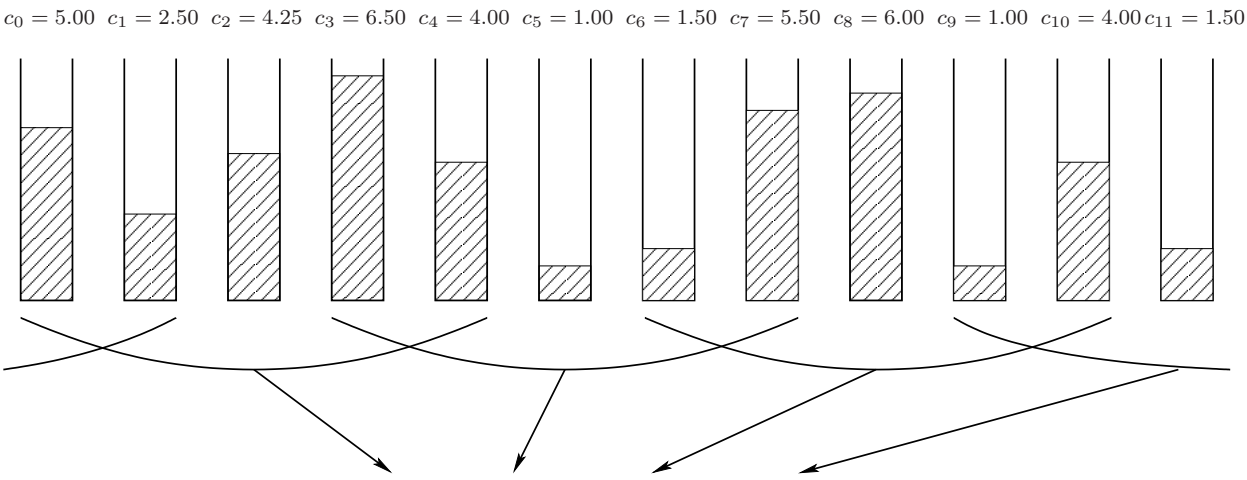

Figure 6.1: Demodulating a $(3,5,12)$-locally rank-modulated signal.

We say a sequence $\mathbf{f}$ of n/s permutations over S_t is (s,t,n)-*LRM realizable* if there exists $\mathbf{c} \in \mathbb{R}^n$ such that $\mathbf{f} = \mathbf{f}_\mathbf{c}$, i.e., it is the demodulated sequence of $\mathbf{c}$ under the (s,t,n)-LRM scheme. Except for the degenerate case of $s = t$, not every sequence is realizable. For example, if $s < t$ and $f_{\mathbf{c}_{i\cdot s,t}}$ is the identity permutation (i.e., $f_{\mathbf{c}_{i\cdot s,t}}(i) = i$), then for all $1 \leq j < n$ we have $c_j < c_{j+1}$, but also $c_n < c_1$, which is impossible.

We denote the set of all (s,t,n)-LRM realizable permutation sequences as $\mathcal{R}(s,t,n)$. In a later part of this section, we show that the number of states representable by an (s,t,n)-LRM scheme, i.e., the size of $\mathcal{R}(s,t,n)$, is roughly $(t \cdot (t-1) \cdot ... \cdot (t-s+1))^{n/s}$ (this fact is also stated in [85]).

While any $\mathbf{f} \in \mathcal{R}(s,t,n)$ may be represented as a sequence of n/s permutations over S_t, a more succinct representation is possible based on the (mixed-radix) factoradic notation system (see [52] for the earliest-known definition, and [45] for a related use): We can represent any permutation $f = [f(0), \ldots, f(t-1)] \in S_t$ with a sequence of digits $d_{t-1}, d_{t-2}, \ldots, d_1, d_0$ (note the reversed order of indices), where $d_{t-1-i} \in \mathbb{Z}_{t-i}$, and d_{t-1-i} counts the number of entries $f(j)$ for $j > i$ which are of value lower than $f(i)$, i.e.,

$$d_{t-1-i} = |\{j > i \mid c_j < c_i\}|.$$

We call d_{t-1} the *most-significant digit* and d_0 the *least-significant digit*. If $f = f_\mathbf{c}$ for some $\mathbf{c} \in \mathbb{R}^t$, then the factoradic representation is easily seen to be equivalent to counting the number of cells to the right of the ith cell which are with lower charge levels.

Continuing with the succinct representation, we now contend that due to the overlap between local views, we can represent each of the local permutations $f_{\mathbf{c}_{i\cdot s,t}}$ using only the s most-significant digits in their factoradic notation. We denote this (partial) representation as $\bar{f}_{\mathbf{c}_{i\cdot s,t}}$ and call it the *condensed factoradic* representation. Accordingly, we define,

$$\bar{\mathbf{f}}_\mathbf{c} = (\bar{f}_{\mathbf{c}_{0,t}}, \bar{f}_{\mathbf{c}_{s,t}}, \bar{f}_{\mathbf{c}_{2s,t}}, \ldots, \bar{f}_{\mathbf{c}_{n-s,t}}),$$

and the set of all such presentations as $\bar{\mathcal{R}}(s,t,n)$. Thus, for example, the configuration of

Figure 6.1 would be represented by $((3,0,1),(4,2,0),(1,2,2),(0,2,0))$. Since throughout the rest of the chapter we shall deal with the condensed factoradic representation only, we omit from now on the term "condensed".

Lemma 6.1: *For all* $1 \leq s \leq t \leq n$,

$$\left|\bar{\mathcal{R}}(s,t,n)\right| \leq |\mathcal{R}(s,t,n)| \leq (t-s)! \cdot \left(\frac{t!}{(t-s)!}\right)^{\frac{n}{s}}.$$

PROOF: As $\bar{\mathcal{R}}(s,t,n) = \{\bar{\mathbf{f}}_{\mathbf{c}} \mid \mathbf{f}_{\mathbf{c}} \in \mathcal{R}(s,t,n)\}$ we have that $\left|\bar{\mathcal{R}}(s,t,n)\right| \leq |\mathcal{R}(s,t,n)|$. For the upper bound, assume we fix the permutation induced by the first $t-s$ cells, where there are $(t-s)!$ ways of doing so. It follows that there are at most $t!/(t-s)!$ ways of choosing $f_{\mathbf{c}_{n-s,t}}$, and then the same bound on the number of ways of choosing $f_{\mathbf{c}_{n-2s,t}}$, and continuing all the way up to $f_{\mathbf{c}_{0,t}}$ we obtain the desired bound. ∎

When $s = t = n$, the (n,n,n)-LRM scheme degenerates into a single permutation from S_n. This was the case in most of the previous works using permutations for modulation purposes. A slightly more general case, $s = t < n$ was discussed by Ferreira *et al.* [24] in the context of permutation trellis codes, where a binary codeword was translated tuple-wise into a sequence of permutations with no overlap between the tuples. An even more general case was defined by Wang *et al.* [85] (though in a slightly different manner where indices are not taken modulo n, i.e., with no wrap-around). In [85], the sequence of permutations was studied under a charge-difference constraint called *bounded rank-modulation*, and mostly with parameters $s = t-1$, i.e., an overlap of one position between adjacent windows.

6.1.2 Gray Codes

A *Gray code*, G, is a sequence of distinct states (codewords), $G = g_0, g_1, \ldots, g_{N-1}$, from an ambient state space, $g_i \in S$, such that adjacent states in the sequence differ by a "small" change. What constitutes a "small" change usually depends on the code's application.

Since we are interested in building Gray codes for flash memory devices with the (s,t,n)-LRM scheme, the ambient space is $\mathcal{R}(s,t,n)$, which is the set of all realizable sequences

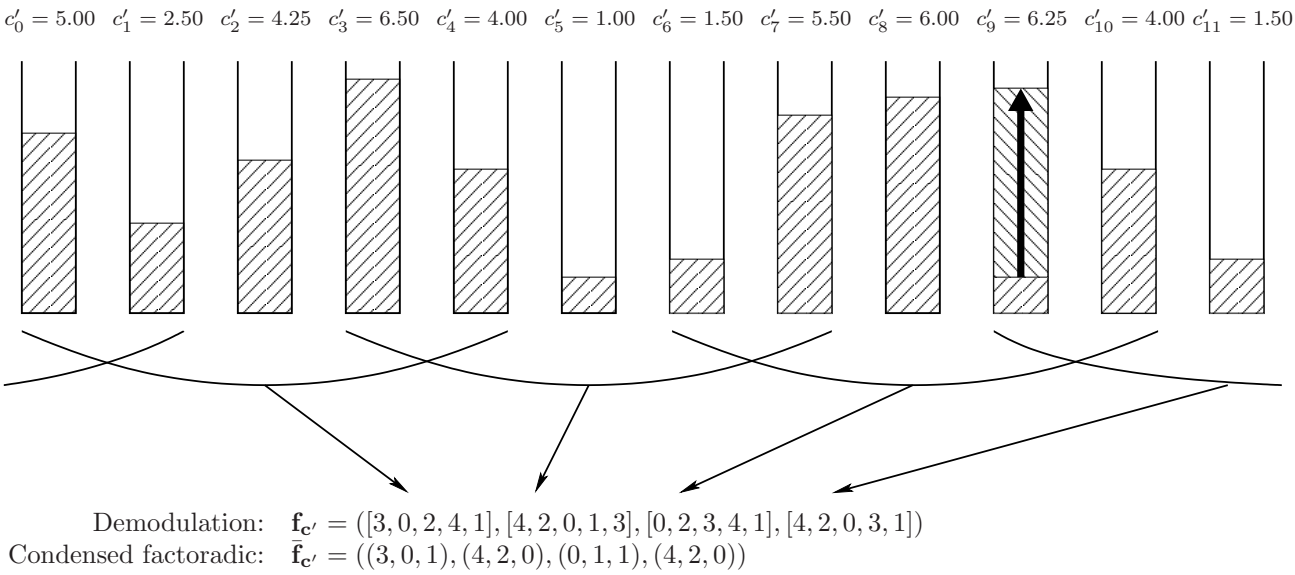

Figure 6.2: A "push-to-the-top" operation performed on the 9th cell.

under (s,t,n)-LRM.

The transition between adjacent states in the Gray code is directly motivated by the flash memory application, and was first described and used in [45]. This transition is the "push-to-the-top" operation, which takes a single flash cell and raises its charge level above all others.

In our case, however, since we are considering a *local* rank-modulation scheme, the "push-to-the-top" operation merely raises the charge level of the selected cell above those cells which are comparable with it.

In Figure 6.2 we see the example signal of Figure 6.1 after a "push-to-the-top" operation performed on the 9th cell. The cells participating with the 9th cell in local permutations are $6, 7, 8, 10, 11, 0$, and 1, i.e., from cell 6 to cell 1 with wrap-around. Thus, the charge level of the 9th cell was pushed above that of cells 6 through 1 (with wrap-around). We do note that the new charge level of the 9th cell is not above that of all other cells since the 3rd cell still has a higher level. However, the 3rd cell is incomparable (i.e., does not participate in a local

permutation) with the 9th cell. Figure 6.2 also shows the demodulation and condensed factoradic representation of the new configuration. By comparing with Figure 6.1, we note that a single "push-to-the-top" operation can change several digits in the demodulated sequence and in the factoradic notation.

We now provide a precise definition of the "push-to-the-top" operation in the local rank-modulation scheme. Assume we have n flash memory cells with charge levels $\mathbf{c} = (c_0, \ldots, c_{n-1}) \in \mathbb{R}^n$. We say cell j is comparable with cell j' if they both participate in some window together. We shall denote these cells as the cells numbered $l(j), l(j)+1, \ldots, r(j)$ where one can easily verify that

$$l(j) = s \left\lceil \frac{j-t+1}{s} \right\rceil \bmod n,$$
$$r(j) = \left(s \left\lfloor \frac{j}{s} \right\rfloor + (t-1) \right) \bmod n.$$

We note that the indices $l(j), l(j)+1, \ldots, r(j)$ should be taken modulo n. Continuing the example of Figure 6.2, for $s = 3$ and $t = 5$ we have $l(8) = 3\lceil (8-5+1)/3 \rceil = 6$ and $r(10) = (3\lfloor 10/3 \rfloor + 5 - 1) \equiv 1 \pmod{12}$, i.e., the left-most cell comparable with cell 8 is cell 6, while the right-most cell comparable with cell 10 is cell 1.

A "push-to-the-top" operation performed on cell j changes the cell levels to $\mathbf{c}' = (c'_0, \ldots, c'_{n-1}) \in \mathbb{R}^n$ defined by

$$c'_i = \begin{cases} c_i & i \neq j, \\ \max\left\{c_{l(i)}, \ldots, c_{r(i)}\right\} + \epsilon & i = j, \end{cases}$$

where $\epsilon > 0$ denotes a small charge difference which is a parameter of the physical charge-placement mechanism. Namely, the charge level of cell j is pushed above the charge levels of cells comparable with it.

We can now move from the physical level to the logical level of the demodulated signal. With the above notation, $\mathbf{c}'$ was achieved from $\mathbf{c}$ by a "push-to-the-top" operation on cell j.

Let $\mathbf{f_c}$ and $\mathbf{f_{c'}}$ stand for the demodulated sequence of permutations from $\mathbf{c}$ and $\mathbf{c'}$, respectively. We then say $\mathbf{f_{c'}}$ was achieved from $\mathbf{f_c}$ by a single "push-to-the-top" operation on cell j. Thus, we define the set of allowed transitions as $T = \{\tau_0, \tau_1, \ldots, \tau_{n-1}\}$, which is a set of functions, $\tau_j : \mathcal{R}(s,t,n) \to \mathcal{R}(s,t,n)$, where τ_j represents a "push-to-the-top" operation performed on the j-th cell.

Definition 6.1: *A* Gray code G for (s,t,n)-LRM (denoted (s,t,n)-LRMGC) *is a sequence of distinct codewords,* $G = g_0, g_1, \ldots, g_{N-1}$, *where* $g_i \in \mathcal{R}(s,t,n)$. *For all* $0 \leq i \leq N-2$, *we further require that* $g_{i+1} = \tau_j(g_i)$ *for some* j. *If* $g_0 = \tau_j(g_{N-1})$ *for some* j, *then we say the code is* cyclic. *We call* N *the* size *of the code, and say* G *is* optimal *if* $N = |\mathcal{R}(s,t,n)|$.

Definition 6.2: *A family of codes,* $\{G_i\}_{i=1}^{\infty}$, *where* G_i *is an* (s,t,n_i)*-LRMGC of size* N_i, $n_{i+1} > n_i$, *is* asymptotically rate-optimal *if*

$$\lim_{i\to\infty} \frac{\log_2 N_i}{\log_2 |\mathcal{R}(s,t,n_i)|} = 1.$$

In this chapter we present two constructions of asymptotically rate-optimal Gray codes. The first construction considers the case of $(1,2,n)$-LRMGC, while the second construction considers the more general case of (s,t,n)-LRMGC. However, the first construction also considers an additional property, in which the codes have a *constant weight*. We discuss this property next.

Let us now assume an optimal setting in which a "push-to-the-top" operation on the j-th cell sets $c'_j = \max\{c_{j-1}, c_{j+1}\} + 1$. A general $(1,2,n)$-LRMGC may result in $c'_j - c_j$ exponential in n, for some transition from g_i to g_{i+1}. The same motivation in the case of (n,n,n)-LRM was discussed in [45], where a balanced variant of Gray codes was constructed to avoid the problem. We present a different variant of Gray codes to address the same issue.

First, for any binary string $v = v_0 v_1 \ldots v_{n-1}$, we call the number of 1's in v the *weight* of v and denote it as $\mathrm{wt}(v)$. We also denote by $S(n,w)$ the set of length-n binary strings of weight w. We now define our variant of Gray codes:

Definition 6.3: *A* constant-weight Gray code for $(1,2,n)$-LRM (denoted $(1,2,n;w)$-LRMGC), *$G = g_0, g_1, \ldots, g_{N-1}$, is a Gray code for $(1,2,n)$-LRM for which $g_i \in S(n,w)$ for all $0 \leq i \leq N-1$.*

Definition 6.4: *Let G be a $(1,2,n;w)$-LRMGC of size N. We define the* rate *of the code as* $\mathrm{R}(G) = \frac{\log_2 N}{n}$. *The* efficiency *of G is defined as* $\mathrm{Eff}(G) = N/\binom{n}{w}$. *If* $\mathrm{Eff}(G) = 1$ *then we say G is* optimal. *If* $\mathrm{Eff}(G) = 1 - o(1)$, *where $o(1)$ denotes a function that tends to 0 as $n \to \infty$, then we say G is* asymptotically optimal.

The transitions between adjacent states in the constant-weight variant take on a very simple form: a window of size 2 in g_i which contains 10 is transformed in g_{i+1} into 01, i.e., "pushing" a logical 1 a single place to the right. Since we are interested in creating cyclic counters, we will be interested in cyclic Gray codes. An example of a cyclic optimal Gray code is given in Table 6.1.

Table 6.1: A cyclic optimal $(1,2,5;2)$-LRMGC (The changed positions are underlined.)

$\underline{1}$	1	0	0	$\underline{0}$
1	$\underline{0}$	$\underline{1}$	0	0
$\underline{0}$	$\underline{1}$	1	0	0
0	1	$\underline{0}$	$\underline{1}$	0
0	$\underline{0}$	$\underline{1}$	1	0
0	0	1	$\underline{0}$	$\underline{1}$
0	0	$\underline{0}$	$\underline{1}$	1
$\underline{1}$	0	0	1	$\underline{0}$
1	0	0	$\underline{0}$	$\underline{1}$
$\underline{0}$	$\underline{1}$	0	0	1

It should be noted that Gray codes with a weaker restriction, allowing a 01 to be changed into 10 and also 10 to be changed back into 01, i.e., a 1 may be pushed either to the right or to the left, have been studied in the past [6, 9, 15, 38, 71].

We can show that under the constant-weight restriction, for any "push-to-the-top" operation,

$$c'_j - c_j \leq \left\lceil \frac{\max\{w, n-w\}}{\min\{w, n-w\}} \right\rceil .$$

This is done by first assuming $2w \leq n$, or else we flip all the bits and reverse the codewords. We will only use integer charge levels, and thus for any codeword, g_i, $\mathrm{wt}(g_i) = w$, we can find a realization by setting $c_{j+1} - c_j = 1$ if $g_{i,j} = 0$, and $c_{j+1} - c_j = -[(n-w)/w]$ if $g_{i,j} = 1$, where $[\cdot]$ denotes either $\lfloor \cdot \rfloor$ or $\lceil \cdot \rceil$.

It is now easily shown by induction that a "push-to-the-top" operation on the j-th cell preserves charge-level differences between adjacent cells and only rearranges their order: by the induction hypothesis, initially we have $c_j - c_{j-1} = -[(n-w)/w]$ and $c_{j+1} - c_j = 1$. The "push-to-the-top" operation sets $c'_j = \max\{c_{j-1}, c_{j+1}\} + 1 = c_{j-1} + 1$ and then $c'_j - c_{j-1} = 1$ and $c_{j+1} - c'_j = -[(n-w)/w]$.

6.2 Constant-Weight Gray Codes for $(1, 2, n)$-LRM

In this section we construct $(1, 2, n)$-LRMGCs with rates asymptotically tending to 1, and weight asymptotically half the length, thus having asymptotically-optimal charge difference between adjacent cells. Our construction has the following intuitive flavor. We start by partitioning the n flash cells into about $\sqrt{n}$ blocks, each block of size about $\sqrt{n}$, treating each block of cells as a single character in a large alphabet, say $\{0, 1, \ldots, t-1\}$ for $t \simeq 2^{\sqrt{n}}$. Roughly speaking, by this operation, we have reduced the problem of finding a Gray code over $\{0,1\}^n$ into an *outer* Gray-like code over $\{0, 1, \ldots, t-1\}^{\sqrt{n}}$. Several Gray codes of rate 1 exist over large alphabets, however, not any outer code will suffice in our setting. Primarily, it is crucial that we may move from state to state in the outer code using our elementary pairwise "push-to-the-top" operations. Moreover, in doing so, we must guarantee that flash cell values obtained between a single representation of the outer codeword and its successor are unique. We achieve these goals using an outer Gray code based on de-Bruijn sequences. In such codes, the location of the character that changes between subsequent codewords over goes a cyclic shift. This cyclic location change between subsequent codewords lends itself very naturally to our cyclic "push-to-the-top" operations. Combining this with additional ideas that guarantee distinct cell values (of constant weight) in transition between outer

codewords, we obtain our construction.

Construction 6.1: *Fix a positive integer k. Let $\{v_0, v_1, \ldots, v_{t-1}\}$ be a set of t distinct binary vectors of length $m+2$ and weight $w+2$ such that the first and last bit of each v_i is 1. We also denote $L = \mathrm{lcm}(k+2, t^k)$.*

The next required ingredient in the construction is a de-Bruijn sequence of order k over the alphabet $\{0, 1, \ldots, t-1\}$. The sequence is of period t^k and we denote it by $s_0, s_1, \ldots, s_{t^k-1}$. We remind the reader that windows of size k in the sequence, i.e., $s_i, s_{i+1}, \ldots, s_{i+k-1}$, with indices taken modulo t^k, are all distinct. Such sequences can always be constructed (for example, see [31]).

We now construct the sequence $g_0, g_1, \ldots, g_{L-1}$ of L binary vectors of length $(k+2)(m+2)$ and weight $(k+1)(w+2)$. Each vector is formed by a concatenation of $k+2$ blocks of length

$m+2$ as follows:

$$
\begin{array}{rccccccc}
g_0 = & v_{s_k} & v_{s_{k-1}} & \dots & v_{s_1} & v_{s_0} & \mathbf{0} \\
g_1 = & v_{s_k} & v_{s_{k-1}} & \dots & v_{s_1} & \mathbf{0} & v_{s_{k+1}} \\
g_2 = & v_{s_k} & v_{s_{k-1}} & \dots & \mathbf{0} & v_{s_{k+2}} & v_{s_{k+1}} \\
 & & & \vdots & & & \\
g_k = & v_{s_k} & \mathbf{0} & \dots & v_{s_{k+3}} & v_{s_{k+2}} & v_{s_{k+1}} \\
g_{k+1} = & \mathbf{0} & v_{s_{2k+1}} & \dots & v_{s_{k+3}} & v_{s_{k+2}} & v_{s_{k+1}} \\
g_{k+2} = & v_{s_{2k+2}} & v_{s_{2k+1}} & \dots & v_{s_{k+3}} & v_{s_{k+2}} & \mathbf{0} \\
g_{k+3} = & v_{s_{2k+2}} & v_{s_{2k+1}} & \dots & v_{s_{k+3}} & \mathbf{0} & v_{s_{2k+3}} \\
g_{k+4} = & v_{s_{2k+2}} & v_{s_{2k+1}} & \dots & \mathbf{0} & v_{s_{2k+4}} & v_{s_{2k+3}} \\
 & & & \vdots & & & \\
g_{2k+2} = & v_{s_{2k+2}} & \mathbf{0} & \dots & v_{s_{2k+5}} & v_{s_{2k+4}} & v_{s_{2k+3}} \\
g_{2k+3} = & \mathbf{0} & v_{s_{3k+3}} & \dots & v_{s_{2k+5}} & v_{s_{2k+4}} & v_{s_{2k+3}} \\
 & & & \vdots & & & \\
g_{L-k-2} = & v_{s_{L-2}} & v_{s_{L-3}} & \dots & v_{s_{L-k-1}} & v_{s_{L-k-2}} & \mathbf{0} \\
g_{L-k-1} = & v_{s_{L-2}} & v_{s_{L-3}} & \dots & v_{s_{L-k-1}} & \mathbf{0} & v_{s_{L-1}} \\
g_{L-k} = & v_{s_{L-2}} & v_{s_{L-3}} & \dots & \mathbf{0} & v_{s_0} & v_{s_{L-1}} \\
 & & & \vdots & & & \\
g_{L-2} = & v_{s_{L-2}} & \mathbf{0} & \dots & v_{s_1} & v_{s_0} & v_{s_{L-1}} \\
g_{L-1} = & \mathbf{0} & v_{s_{k-1}} & \dots & v_{s_1} & v_{s_0} & v_{s_{L-1}}
\end{array}
$$

where $\mathbf{0}$ denotes the all-zero vector of length $m+2$, and the sub-indices of s are taken modulo t^k.

We call $g_0, g_1, \dots, g_{L-1}$ the anchor vectors. *We note that between anchors g_i and g_{i+1} the block v_{s_i} moves $m+2$ positions to the right (with wrap-around) and is changed to the block $v_{s_{i+k+1}}$.*

Finally, between any two anchors, g_i and g_{i+1}, a sequence of vectors called auxiliary vectors *and denoted $g_i^0, g_i^1, \dots, g_i^{\ell_i}$, is formed in the following way: The only allowed transition*

is a 10 *changed into a* 01*. First the rightmost* 1 *in the block* v_{s_i} *is moved to the right, step by step, to the position of the rightmost* 1 *in* $v_{s_{i+k+1}}$*. The process then repeats with a sequence of transitions moving the second-from-right* 1 *in* v_{s_i} *to the position of the second-from-right* 1 *in* $v_{s_{i+k+1}}$*, and so on, until* v_{s_i} *is moved one block to the right and changed into* $v_{s_{i+k+1}}$ *(see Example* 6.1*). The resulting list of anchor vectors and, in between them, auxiliary vectors, is the constructed code.*

Example 6.1: *Let us take a very simple case of* $k = 1$*,* $m = 3$*,* $w = 2$*, and* $t = 3$*, with* $s_0 = 0$*,* $s_1 = 1$*, and* $s_2 = 2$*, and then* $v_0 = 11101$*,* $v_1 = 11011$*, and* $v_2 = 10111$*. The list of anchors is*

$$\begin{aligned} g_0 = &\quad 11011 \quad 11101 \quad 00000 \\ g_1 = &\quad 11011 \quad 00000 \quad 10111 \\ g_2 = &\quad 00000 \quad 11101 \quad 10111 \end{aligned}$$

and, for example, the transition between g_0 *and* g_1 *is shown in Table* 6.2 *(the changed positions are underlined).*

Theorem 6.1: *The code constructed in Construction* 6.1 *is a cyclic* $(1, 2, (k+2)(m+2); (k+1)(w+2))$*-LRMGC of size*

$$N = L(w+2)(m+2) = \mathrm{lcm}(t^k, k+2) \cdot (w+2) \cdot (m+2).$$

The proof of Theorem 6.1 requires the following definition:

Definition 6.5: *For any* $v = v_0 v_1 \dots v_{n-1} \in S(n, w)$*, we define the first moment of* v *as*

$$\chi(v) = \sum_{j=0}^{n-1} j \cdot v_j \tag{6.2}$$

and the color *of* v *as* $\chi(v) \bmod n$*.*

Proof (of Theorem 6.1): That the code contains only valid transitions is evident by the construction method. We need to show that all the constructed codewords are distinct,

Table 6.2: The transitions between anchors in Example 6.1.

$$
\begin{array}{rcccc}
g_0 = & 11011 & 11101 & 00000 \\
g_0^0 = & 11011 & 1110\underline{0} & \underline{1}0000 \\
g_0^1 = & 11011 & 11100 & \underline{01}000 \\
g_0^2 = & 11011 & 11100 & 0\underline{01}00 \\
g_0^3 = & 11011 & 11100 & 00\underline{01}0 \\
g_0^4 = & 11011 & 11100 & 000\underline{01} \\
g_0^5 = & 11011 & 11\underline{01}0 & 00001 \\
g_0^6 = & 11011 & 110\underline{01} & 00001 \\
g_0^7 = & 11011 & 1100\underline{0} & \underline{1}0001 \\
g_0^8 = & 11011 & 11000 & \underline{01}001 \\
g_0^9 = & 11011 & 11000 & 0\underline{01}01 \\
g_0^{10} = & 11011 & 11000 & 00\underline{01}1 \\
g_0^{11} = & 11011 & 1\underline{01}00 & 00011 \\
g_0^{12} = & 11011 & 10\underline{01}0 & 00011 \\
g_0^{13} = & 11011 & 100\underline{01} & 00011 \\
g_0^{14} = & 11011 & 1000\underline{0} & \underline{1}0011 \\
g_0^{15} = & 11011 & 10000 & \underline{01}011 \\
g_0^{16} = & 11011 & 10000 & 0\underline{01}11 \\
g_0^{17} = & 11011 & \underline{01}000 & 00111 \\
g_0^{18} = & 11011 & 0\underline{01}00 & 00111 \\
g_0^{19} = & 11011 & 00\underline{01}0 & 00111 \\
g_0^{20} = & 11011 & 000\underline{01} & 00111 \\
g_1 = & 11011 & 0000\underline{0} & \underline{1}0111
\end{array}
$$

which we do with the following reasoning: consider some constructed codeword g of length $(k+2)(m+2)$ and weight $(k+1)(w+2)$. Deciding whether g is an anchor is simple, since only anchors have $k+1$ blocks beginning and ending with a 1, and the remaining block a $\mathbf{0}$. By our choice of L, all anchors are distinct since they contain windows of size $k+1$ from a de-Bruijn sequence of order k, each window appearing in $(k+2)/\gcd(k+2, t^k)$ distinct cyclic shifts (which are easily distinguishable by the position of the $\mathbf{0}$ block). It then follows that if g is indeed an anchor it appears only once in the code.

Assume we discover g is an auxiliary vector. Again, by construction, all auxiliary vectors between g_i and g_{i+1} have k fixed blocks. Looking at g, an auxiliary vector, exactly k blocks are of weight $w+2$ while the other two blocks have weight strictly below $w+2$. The blocks of weight $w+2$, by construction, form a window of size k from a de-Bruijn sequence of order k starting at s_i, and so their content and position uniquely identify between which two anchors g lies.

Finally, all the auxiliary vectors between adjacent anchors g_i and g_{i+1} are easily seen to be distinct. Thus, given a codeword g from the constructed code, there is exactly one position in the sequence of generated codewords which equals g, and so all generated codewords are distinct.

To complete the proof we need to calculate the size N. There are exactly L anchors. Given an anchor g_i, the number of steps in the transition to g_{i+1} may be readily verified to be $(w+2)(m+2)+\chi(g_{i+1})-\chi(g_i)$, where $\chi(\cdot)$ is the first moment function defined in (6.2). Thus,

$$\begin{aligned} N &= \sum_{i=0}^{L-1} \left((w+2)(m+2) + \chi(g_{i+1}) - \chi(g_i)\right) \\ &= L(w+2)(m+2) \end{aligned}$$

as claimed. As a final note, the choice of L is easily seen to ensure the resulting code is cyclic. ∎

We mention in passing that the proof of Theorem 6.1 hints at efficient encoding and decoding procedures, provided other efficient encoding and decoding procedures exist for de-Bruijn sequences. Examples of such procedures may be found in [61, 84].

We now turn to show the main claim of the section.

Corollary 6.1: *There exists an infinite family* $\{G_i\}_{i=1}^{\infty}$ *of cyclic* $(1, 2, n_i; w_i)$*-LRMGCs,* $n_{i+1} > n_i$ *for all* i*, for which* $\lim_{i\to\infty} \mathrm{R}(G_i) = 1$*, and* $\lim_{i\to\infty} \frac{w_i}{n_i} = \frac{1}{2}$.

PROOF: For the code G_i, set $w = i$, and $k = m = 2i$ (i.e., $n_i = (2i+2)^2$ and $w_i = (2i+1)(i+2)$) and apply Theorem 6.1 with $t = \binom{2i}{i}$. The size, N_i, of the code G_i, is bounded by

$$\binom{2i}{i}^{2i}(i+2)(2i+2) \leq N_i \leq \binom{2i}{i}^{2i}(i+2)(2i+2)^2$$

since

$$\binom{2i}{i}^{2i} \leq \mathrm{lcm}\left(\binom{2i}{i}^{2i}, 2i+2\right) \leq \binom{2i}{i}^{2i}(2i+2).$$

It is well known (see for example [57], p. 309) that for any $0 < \lambda < 1$, assuming $\lambda\ell$ is an integer,

$$\frac{1}{\sqrt{8\ell\lambda(1-\lambda)}}2^{\ell H(\lambda)} \leq \binom{\ell}{\lambda\ell} \leq \frac{1}{\sqrt{2\ell\lambda(1-\lambda)}}2^{\ell H(\lambda)}$$

where $H(\cdot)$ is the binary entropy function. Since $H(1/2) = 1$, it now easily follows that

$$\lim_{i\to\infty} \mathrm{R}(G_i) = 1, \qquad \lim_{i\to\infty} \frac{w_i}{n_i} = \frac{1}{2}.$$

■

If needed, we can achieve lower asymptotic rates by setting $w = \lambda m$ for some rational $0 < \lambda < 1$, $\lambda \neq 1/2$.

6.3 Gray Codes for (s, t, n)-LRM

In this section we present efficiently encodable and decodable asymptotically-rate-optimal Gray codes for (s, t, n)-LRM. A rough description of the proposed construction follows.

First we partition the n cells into m blocks, each containing n/m cells. To simplify the presentation we set $n = m^2$, implying that we have m blocks, each of size m. Denote the cells in block i by $\mathbf{c}_i$. For each block $\mathbf{c}_i$ we will use the factoradic representation $\bar{\mathbf{f}}_{\mathbf{c}_i}$ to represent permutations, without wrap-around at the block level (the wrap-around is only for the entire codeword). Namely, each and every block can be thought of as an element of an alphabet $\Sigma = \{v_0, \ldots, v_{V-1}\}$ of a size denoted by V.

Now, consider any De-Bruijn sequence S of order $m-1$ over Σ (of period V^{m-1}). Namely, S will consist of a sequence of V^{m-1} elements $v_{s_0}, v_{s_1}, \ldots, v_{s_{V^{m-1}-1}}$ over Σ such that the subsequences $v_{s_i}, \ldots, v_{s_{i+m-2}}$ of S *cover* all $(m-1)$-tuples of Σ exactly once, sub-indices of s are taken modulo V^{m-1}. We shall conveniently choose $\Sigma = \mathbb{Z}_V$. Such sequences S exist, e.g., [31].

The construction of the Gray code G will have two phases. First we construct so-called *anchor* elements in G, denoted as $\bar{G} = \{g_0, \ldots, g_{L-1}\}$. The elements of $\bar{G}$ will consist of a cyclic Gray code over Σ^m. That is, the difference between each g_i and g_{i+1} in $\bar{G}$ will be in only one out of the m characters (from Σ) in g_i. Specifically, the code $\bar{G}$ will be derived from the De-Bruijn sequence S as follows: we set g_0 to be the first m elements of S, and in the transition from g_i to g_{i+1} we change v_{s_i} to $v_{s_{i+m}}$. The code $\bar{G}$ is detailed below:

$$
\begin{aligned}
g_0 &= v_{s_{m-1}} \; v_{s_{m-2}} \; \ldots \; v_{s_1} \; \underline{v_{s_0}} \\
g_1 &= v_{s_{m-1}} \; v_{s_{m-2}} \; \ldots \; \underline{v_{s_1}} \; v_{s_m} \\
g_2 &= v_{s_{m-1}} \; v_{s_{m-2}} \; \ldots \; v_{s_{m+1}} \; v_{s_m} \\
&\vdots \\
g_{L-2} &= v_{s_{L-1}} \; \underline{v_{s_{L-2}}} \; \ldots \; v_{s_1} \; v_{s_0} \\
g_{L-1} &= \underline{v_{s_{L-1}}} \; v_{s_{m-2}} \; \ldots \; v_{s_1} \; v_{s_0},
\end{aligned}
$$

where $L = \mathrm{lcm}(m, V^{m-1})$, the sub-indices of s are taken modulo V^{m-1}, and the underline is an imaginary marking distinguishing the block which is about to change.

With the imaginary marking of the underline, the code $\bar{G}$ is clearly a Gray code over Σ^m due to the properties of the De-Bruijn sequence S. However $\bar{G}$ is not a Gray code over

$\mathcal{R}(s,t,n)$, as the transitions between the anchors g_i and g_{i+1} require changing the entries of an entire block, which may involve many push-to-the-top operations. We thus refine $\bar{G}$ by adding additional elements between each pair of adjacent anchors from $\bar{G}$ that allow us to move from the block configuration in g_i to that in g_{i+1} by a series of push-to-the-top operations.

Each block is eventually represented in factoradic notation by a sequence of digits. For the construction to work, it is crucial to be able to identify, for each element in $\bar{G}$, which block is in the process of being changed. To this end, and for other technical reasons, we will add some auxiliary digits to each block (by adding auxiliary flash memory cells). These digits are referred to as *non-information* digits. Loosely speaking, in each block of size m the last $t+1$ digits (approximately[1]) will be non information digits. The last 2 digits will be used to mark which block is being currently changed, while the $t-1$ digits before them will act as a *buffer zone* that allows the successful transformation between anchors. In our setting, the $t+1$ non-information digits will be negligible with respect to the remaining $m-(t+1)$ information digits, allowing the code $\bar{G}$ to have asymptotically optimal rate.

Construction 6.2: *We consider the (s,t,n)-LRM scheme, where*

$$n = m^2, \qquad m \geq t+2, \qquad s \mid m.$$

Let $\{v_0, v_1, \ldots, v_{V-1}\}$ be a set of V distinct mixed-radix vectors of length m. Each vector v_i is representing m/s local permutations, where each permutation is represented by the s most-significant digits of its factoradic notation. Therefore,

$$v_i \in \left(\mathbb{Z}_t \times \mathbb{Z}_{t-1} \times \cdots \times \mathbb{Z}_{t-(s-1)}\right)^{m/s}.$$

Let us denote

$$\delta = \left\lceil \frac{t+2}{s} \right\rceil - 1.$$

[1]For the exact value see Construction 6.2.

The values of the last $s\delta$ digits (that represent the last δ local permutations) of each v_i do not play a role in the representation of the stored data and are called non-information digits. *By abuse of notation, two vectors agreeing on the first $m - s\delta$ digits will be said to represent the same value. Furthermore, when a block is said to represent some value v_i, we mean that its first $m - s\delta$ digits agree with those of v_i. Therefore, we set*

$$V = \left(\frac{t!}{(t-s)!}\right)^{\frac{m}{s}-\delta}.$$

We also denote $L = \mathrm{lcm}(m, V^{m-1})$.

Let S be a De-Bruijn sequence of order $m-1$ over the alphabet $\mathbb{Z}_V$,

$$S = s_0, s_1, \ldots, s_{V^{m-1}-1},$$

i.e., S is of length V^{m-1} and $s_i \in \mathbb{Z}_V$. The Gray code $\bar{G}$ of anchor vectors is a sequence

$$\bar{G} = g_0, g_1, \ldots, g_{L-1}$$

of L mixed-radix vectors of length $n = m^2$. Each vector is formed by a concatenation of m blocks of length m. For $k \in [m]$, we say that block k corresponds to the cells with indices $km, km+1, \ldots, (k+1)m-1$. We set g_0 to be the concatenation of the first m elements of S, such that for each $k \in [m]$, block k represent the vector $v_{s_{m-1-k}}$:

$$g_0 = v_{s_{m-1}} v_{s_{m-2}} \ldots v_{s_1} v_{s_0}.$$

Between the anchors g_i and g_{i+1}, the block that represents the vector v_{s_i} is transformed into

the vector $v_{s_{i+m}}$. The resulting Gray code $\bar{G}$ of anchor vectors is therefore

$$\begin{aligned}
g_0 &= v_{s_{m-1}} \quad v_{s_{m-2}} \quad \dots \quad v_{s_1} \quad \underline{v_{s_0}} \\
g_1 &= v_{s_{m-1}} \quad v_{s_{m-2}} \quad \dots \quad \underline{v_{s_1}} \quad v_{s_m} \\
g_2 &= v_{s_{m-1}} \quad v_{s_{m-2}} \quad \dots \quad v_{s_{m+1}} \quad v_{s_m} \\
&\qquad\qquad\qquad \vdots \\
g_{L-2} &= v_{s_{L-1}} \quad \underline{v_{s_{L-2}}} \quad \dots \quad v_{s_1} \quad v_{s_0} \\
g_{L-1} &= \underline{v_{s_{L-1}}} \quad v_{s_{m-2}} \quad \dots \quad v_{s_1} \quad v_{s_0},
\end{aligned}$$

where the underline denotes the block that is about to change in the transition to the following anchor vector.

Within each of the m blocks comprising any single anchor, the $(m-2)$nd digit (the next-to-last digit – a non-information digit) corresponds to a cell that is pushed-to-the-top in all blocks except for the "underlined" block (i.e., the block which is about to change). For brevity, we call this digit the underlined digit*. In the underlined block, the $(m-1)$th digit is pushed-to-the-top. All remaining non-information digits are initialized to be of value 0.*

Between any two anchors, g_i and g_{i+1}, a sequence of vectors called auxiliary vectors *and denoted $g_i^0, g_i^1, \dots, g_i^{\ell_i}$, is formed by a sequence of push-to-the-top operations on the cells of the changing block. The auxiliary vectors are determined by Algorithm* 6.1 *described shortly. Thus, the entire Gray code G constructed is given by the sequence*

$$\begin{aligned}
&g_0, \quad g_0^0, \quad g_0^1, \quad \dots \quad g_0^{\ell_0}, \\
&g_1, \quad g_1^0, \quad g_1^1, \quad \dots \quad g_1^{\ell_1}, \\
&\vdots \\
&g_{L-1}, \quad g_{L-1}^0, \quad g_{L-1}^1, \quad \dots \quad g_{L-1}^{\ell_{L-1}}.
\end{aligned}$$

In what follows we present Algorithm 6.1 that specifies the sequence $g_i^0, g_i^1, \dots, g_i^{\ell_i}$ that allows us to move from anchor state g_i to state g_{i+1}. As g_i and g_{i+1} differ only in a single block (and this block is changed from representing the value v_{s_i} to $v_{s_{i+m}}$), it holds that g_i^j

and $g_i^{j'}$ will differ only in the block in which g_i and g_{i+1} differ. Thus, it suffices to define in Algorithm 6.1 how to change a block of length m with cell values that represent v_{s_i} into a block that represents $v_{s_{i+m}}$ using push-to-the-top operations. However, we call the attention of the reader to the fact that while the change in represented value affects only one block (denoted as block k), for administrative reasons, in block $k-1$ (modulo m) we also push a *non-information* cell. The inputs of Algorithm 6.1 are the vector $v_{s_{i+m}}$ and the corresponding cell configuration $(c_{km}, c_{km+1}, \ldots, c_{(k+1)m-1})$, and its output is the *order* in which those cells are pushed in order to transform the vector represented in block k into $v_{s_{i+m}}$. Algorithm 6.1 also ensures that the underlined digits of blocks k and $k-1$ (modulo m) of an auxiliary vector are both not of maximal value (among the cells with which they share a window). This allows to identify whether the vector is an anchor vector or an auxiliary one. Assuming that

$$(r_0, r_1, \ldots, r_{m-1}) \in \left(\mathbb{Z}_t \times \mathbb{Z}_{t-1} \times \cdots \times \mathbb{Z}_{t-(s-1)}\right)^{m/s}$$

represents the value v_ℓ, then we say that the jth digit of v_ℓ is

$$v_\ell(j) = \begin{cases} r_j & 0 \le j < m - s\delta \\ 0 & \text{otherwise,} \end{cases}$$

i.e., we always force a 0 for the non-information digits. Finally, we restrict $l(\cdot)$ and $r(\cdot)$ by defining

$$l'(j) = \begin{cases} l(j) & 0 \le l(j) \le m-3 \\ 0 & \text{otherwise} \end{cases}$$

$$r'(j) = \begin{cases} r(j) & 0 \le r(j) \le m-3 \\ m-3 & \text{otherwise.} \end{cases}$$

Algorithm 6.1 is strongly based on the factoradic representation of $v_{s_{i+m}}$. Let $v_{s_{i+m}}(j)$ be the jth entry in this representation. Namely, if $\mathbf{c} = (c_0, \ldots, c_{m-1})$ is a cell configuration

Algorithm 6.1 Transform block k of configuration $\mathbf{f_c}$ from v_{s_i} to $v_{s_{i+m}}$

Input: current cell configuration $\mathbf{f_c}$, block number k, new block value $v_{s_{i+m}}$
Output: new cell configuration $\mathbf{f_{c'}}$
Push cell $m-1$ (the last cell) in block $k-1$ (modulo m).
$a_j \Leftarrow 0$ for all $j = 0, 1, \ldots, m-3$
$j \Leftarrow 0$
repeat
  if $v_{s_{i+m}}(j) = \sum_{i=j+1}^{r'(j)} a_i$ and $a_j = 0$ **then**
    if $c_{km+j} \leq \max_{l'(j) \leq j' \leq r'(j), a_{j'}=1} c_{km+j'}$ **then**
      Push the jth cell in block k (the cell in position $km+j$).
    end if
    $a_j \Leftarrow 1$
    $j \Leftarrow l'(j)$
  else
    $j \Leftarrow j+1$
  end if
until $j = m-2$
Push cell $m-2$ (the next-to-last cell) in block k.
Output the resulting cell configuration $\mathbf{f_{c'}}$.

that corresponds to $v_{s_{i+m}}$, then for each index $j \in [m]$ we let $v_{s_{i+m}}(j)$ denote the number of entries in the window corresponding to j that are in a larger position than j and are of value lower than c_j. Roughly speaking, to obtain such a configuration $\mathbf{c}$, Algorithm 6.1, for $j \in [m]$, *marks* each cell c_j in $\mathbf{c}$ after exactly $v_{s_{i+m}}(j)$ cells in positions larger than j (and participating in the window corresponding to j) have been marked. Here, in order to keep track of which cells should not be pushed anymore, we save an array of bits a_j for each cell in the block (initialized to 0), indicating whether the cell c_j should not be pushed anymore. If $a_j = 1$, we say that cell j is marked. Furthermore, when the cell is marked, it is also pushed-to-the-top if its value is lower than that of a cell that shares a window with it and is already marked (the value comparison can be inferred from $\mathbf{f_c}$). Since each time a cell is changed it is pushed-to-the-top, this will ensure that the resulting cell configuration $\mathbf{c}$ will have a factoradic representation corresponding to $v_{s_{i+m}}$.

We note that in order to be able to decode a state, we need to have some way of knowing

which block is being currently changed, i.e., the imaginary underline in the anchor. We use the last two cells of each block for that purpose as described in the example below.

Example 6.2: *Let us consider the case of* $(1, 2, 16)$*-LRM, i.e.,* $s = 1$*,* $t = 2$*,* $m = 4$*, and* $n = m^2 = 16$*. Thus,*

$$\delta = \left\lceil \frac{t+2}{s} \right\rceil - 1 = \left\lceil \frac{2+2}{1} \right\rceil - 1 = 3,$$

and so in each block, the last $s\delta = 3$ *digits are non-information digits, leaving only the first digit in each block to be an information digit.*

According to the construction we set

$$V = \left(\frac{t!}{(t-s)!} \right)^{\frac{m}{s}-\delta} = \left(\frac{2!}{(2-1)!} \right)^{\frac{4}{1}-3} = 2.$$

We therefore take a De-Bruijn sequence of order 3 *and alphabet of size* 2*,*

$$S = 0, 0, 0, 1, 0, 1, 1, 1.$$

The list of anchors is

$$\begin{aligned}
g_0 &= \mathbf{1}010 \quad \mathbf{0}010 \quad \mathbf{0}010 \quad \underline{\mathbf{0}000} \\
g_1 &= \mathbf{1}010 \quad \mathbf{0}010 \quad \underline{\mathbf{0}000} \quad \mathbf{0}010 \\
g_2 &= \mathbf{1}010 \quad \underline{\mathbf{0}000} \quad \mathbf{1}010 \quad \mathbf{0}010 \\
g_3 &= \underline{\mathbf{1}000} \quad \mathbf{1}010 \quad \mathbf{1}010 \quad \mathbf{0}010 \\
g_4 &= \mathbf{1}010 \quad \mathbf{1}010 \quad \mathbf{1}010 \quad \underline{\mathbf{0}000} \\
g_5 &= \mathbf{1}010 \quad \mathbf{1}010 \quad \underline{\mathbf{1}000} \quad \mathbf{0}010 \\
g_6 &= \mathbf{1}010 \quad \underline{\mathbf{1}000} \quad \mathbf{0}010 \quad \mathbf{0}010 \\
g_7 &= \underline{\mathbf{1}000} \quad \mathbf{0}010 \quad \mathbf{0}010 \quad \mathbf{0}010
\end{aligned}$$

The bold bit (the leftmost bit in each group of four) denotes the information bit, while the rest are non-information bits. The underlined vectors are easily recognizable by the next-to-rightmost (next-to-last) bit being 0*.*

Notice that in this example the information bit is dominated in size by the remaining bits

of each block. This is an artifact of our example in which we take n to be small. For large values of n the overhead in each block is negligible with respect to the information bits.

As an example, the transition between g_1 and g_2 is (the changed positions are underlined):

$$\begin{aligned} g_1 &= 1010 \quad 0010 \quad 0000 \quad 0010 \\ g_1^0 &= 1010 \quad 00\underline{0}1 \quad 0000 \quad 0010 \\ g_1^1 &= 1010 \quad 000\underline{0} \quad \underline{1}000 \quad 0010 \\ g_2 &= 1010 \quad 0000 \quad 10\underline{1}0 \quad 0010 \end{aligned}$$

In this example, three cells are pushed. First, the last cell in block number 1 *is pushed, according to the first line of Algorithm* 6.1*. The push affects the value of the last two digits of that block, and in such, signifies that the new vector is not an anchor. Next, the first cell of block* 2 *is pushed, affecting the value of both the last digit of block* 1 *and the first digit of block* 2*. Note that the last digit of block* 1 *is not an information bit, and it has no meaning in the decoding of the Gray code. Finally, the next-to-last bit of block* 2 *is pushed, signifying that the new vector is an anchor.*

We now address the analysis of Algorithm 6.1.

Lemma 6.2: *Assuming the position of the underlined digit is known, all anchors used in Construction* 6.2 *are distinct.*

PROOF: The proof follows directly from the properties of the De-Bruijn sequence S and the fact that we are taking L to be the $\mathrm{lcm}(m, V^{m-1})$. ∎

Lemma 6.3: *Algorithm* 6.1 *maintains the correctness of the underlined digit in anchors (that is, the digit signifies correctly which block is about to change). In addition, between any two adjacent anchors, Algorithm* 6.1 *guarantees that the underlined digits of the changing block (block k) and its predecessor (block $(k-1) \bmod m$) are both not of maximal value (among the cells with which they share a window).*

PROOF: The proof is by induction. The base case follows from the construction of the first anchor element g_0. Assume g_i satisfies the inductive claim. When applying Algorithm 6.1 to move from anchor g_i to g_{i+1}, we start by pushing the last cell of block $k-1$. This implies that the value of the underlined cell in both block k and block $k-1$ (modulo m) are now not maximal. This state of affairs remains until the end of Algorithm 6.1, in which we push the next-to-the last cell in the changed block (block k). At that point in time, the underlined cell in the changed block obtains its maximal value, while block $k-1$ (that is to be changed in the next application of Algorithm 6.1) is of non-maximal value. All the other underlined cells remain unchanged throughout the execution of Algorithm 6.1. ■

Lemma 6.4: *All words in the code G are distinct.*

PROOF: First, remember that the words in G are permutation sequences, while the construction is using the factoradic notation. However, different factoradic vectors always correspond to different permutation sequences, and thus it is enough to show that the factoradic vectors are distinct. Next, by Lemmas 6.2 and 6.3, we know that all of the anchors in Construction 6.2 are distinct. It is left to show that adding the auxiliary vectors resulting from Algorithm 6.1, G remains a Gray code. It is easily seen that a non-anchor codeword can never be mistaken for an anchor, and that due to the De-Bruijn sequence, two auxiliary vectors g_i^j and $g_{i'}^{j'}$ can never be the same when $i \neq i'$.

Thus, it suffices to focus only on the block being changed. In order to show that every word generated by Algorithm 6.1 in a single execution is distinct, we will show that every cell configuration we encounter will never be visited again. Specifically, given a configuration, we let j be the next cell that will be pushed. Since cell j is going to be pushed, there exists a cell j' such that $c_j < c_{j'}$ and $a_{j'} = 1$. After the push, $c_j > c_{j'}$. But since $a_{j'} = 1$, cell j' will not be pushed anymore, and thus in all future configurations cell j will be higher than cell j'. Therefore all future configurations will differ from the initial one. ■

Lemma 6.5: *Algorithm* 6.1 *terminates, and when it does, all of the cells are marked exactly once.*

PROOF: The index j is incremented by 1 in the algorithm's loop, unless a cell is marked. Since a cell cannot be marked more than once, the algorithm must terminate.

For each non-information digit index $j' \in \{m - s\delta, \ldots, m - 3\}$, we forced $v_{s_{i+m}}(j') = 0$, and therefore each of those cells is marked the first time that $j = j'$. Now we assume by induction that for each $j' \leq m - s\delta$, all of the cells with indices l, $j' \leq l \leq m - 3$, are marked before the algorithm terminates.

The base case, $j' = m - s\delta$, was already proved above. For the induction step, by the induction assumption, we know that all the cells in $\{j', \ldots, m - 3\}$ are eventually marked, and in particular, the cells in $\{j', \ldots, r'(j' - 1)\}$ are eventually marked. At the point where exactly $v_{s_{i+m}}(j' - 1)$ of them are marked, the index j in the algorithm is guaranteed to be lowered below $j' - 1$, and so, cell $j' - 1$ will be marked the next time it is visited. Since the algorithm never marks a cell more than once, the claim is proved. ∎

Theorem 6.2: *Algorithm* 6.1 *changes a block representing* v_{s_i} *into a block representing* $v_{s_{i+m}}$.

PROOF: When cell j is being marked, exactly $v_{s_{i+m}}(j)$ cells from $\{j + 1, \ldots, r'(j)\}$ are marked with $a_j = 1$, and thus will not be pushed anymore. The rest will be pushed after and above it, and therefore its rank is exactly $v_{s_{i+m}}(j)$, as desired. ∎

Lemma 6.6: *The time complexity of Algorithm* 6.1 *is* $O(tm)$.

PROOF: Each cell is visited by the algorithm at most t times, once during the first visit of the algorithm, and once following each of the $t - 1$ cells immediately to its right being pushed. Since each cell is pushed at most once, a full execution of the algorithm takes $O(tm)$ steps. ∎

Combining all of the observations up to now, we are able to summarize with the following theorem for G from Construction 6.2.

Theorem 6.3: *The code* G *from Construction* 6.2 *is a Gray code of size at least* L.

Corollary 6.2: *For all constants $1 \leq s < t$, there exists an asymptotically-rate-optimal family of codes, $\{G_i\}_{i=t+2}^{\infty}$, where G_i is an (s,t,n_i)-LRMGC of size N_i, $n_{i+1} > n_i$, with*

$$\lim_{i\to\infty} \frac{\log_2 N_i}{\log_2 |\mathcal{R}(s,t,n_i)|} = 1.$$

PROOF: We set $n_i = s^2 i^2$ for all $i \geq t+2$, and define L_i and V_i according to L and V in Construction 6.2 (where i is the index of the code in the family of codes). Then $N_i \geq L_i \geq V_i^{si-1}$. It follows that

$$\begin{aligned}
&\lim_{i\to\infty} \frac{\log_2 N_i}{\log_2 |\mathcal{R}(s,t,n_i)|} \geq \\
&\quad \geq \lim_{i\to\infty} \frac{(si-1)\log_2 V_i}{\log_2 \left((t-s)! \cdot \left(\frac{t!}{(t-s)!} \right)^{si^2} \right)} \\
&\quad = \lim_{i\to\infty} \frac{(si-1)\log_2 \left(\frac{t!}{(t-s)!} \right)^{i-\left\lceil \frac{t+2}{s} \right\rceil+1}}{\log_2 \left((t-s)! \cdot \left(\frac{t!}{(t-s)!} \right)^{si^2} \right)} \\
&\quad = 1.
\end{aligned}$$

■

In order to use the Gray codes of Construction 6.2, we need to have a way to associate the codewords to their indices. An encoding method should allow one to identify with any index i the corresponding codeword g_i, and a decoding method should offer the reverse functionality. In addition, it is useful to also have a next-step method that calculates the next word in the code.

A next-step method for Construction 6.2 is straightforward to define. Given a word g_i, consider first the case that it is an anchor vector. The next-step algorithm first identifies the block that is about to be changed, according to its non-maximal underlined digit. Similarly, if g_i is an auxiliary vector, the next-state algorithm identifies the block that is currently being changed. With that information at hand, the algorithm continues and finds the De

Bruijn sub-sequence represented by g_i. According to the sub-sequence, the algorithm now finds the next symbol in the De-Bruijn sequence. Efficient next-state algorithms for De-Bruijn sequences, as well as encoding and decoding methods, are described in [61, 84]. With the knowledge of the next De-Bruijn symbol, the next-state algorithm runs Algorithm 6.1 to find the cell that should be pushed next, and accordingly, the next state in the Gray code.

It is natural to try to apply the same idea for encoding and decoding as well. However, there is an obstacle that makes it less straightforward in this case. Consider the decoding function for example. By the same means as before, we can identify the index in the De-Bruijn sequence quickly. But in order to use it for identifying the index in the Gray code, we would need to know the *distance* between each pair of adjacent anchors. The problem is that this distance that is determined by the number of pushed cells in Algorithm 6.1 is not constant. Therefore, we would need to calculate it for each of the previous symbols in the De-Bruijn sequence. Since the length of the De-Bruijn sequence is exponential in m, this method would be inefficient. We note that this problem also applies to the construction that was presented in [19].

To tackle this obstacle, we suggest a slight modification to Construction 6.2, that allows for efficient encoding and decoding in the manner described above. The main purpose of the modification is to make the distance between the anchors constant. When this distance is constant, the decoder can simply multiply it with the De-Bruijn index, and find the index of the nearest anchor smaller than g_i. From there, applying Algorithm 6.1 completes the decoding efficiently. A similar observation holds for the encoder.

To make the distance between the anchors constant, we take the approach of using an additional counter. We aim to make the distance to always be $m + 2$, i.e., exactly $m + 1$ auxiliary vectors between adjacent anchors. Since Algorithm 6.1 creates between 1 to $m - 1$ auxiliary vectors between anchors (not including the anchors), we use the counter to create between 2 to m additional auxiliary vectors. Therefore, the counter should be able to count from 0 up to $m - 1$, and needs to be capable of resetting, to prepare for usage in the next block. To implement the counter, we add another block to Construction 6.2.

We start by describing a simple construction for a counter which we shall later append to Construction 6.2. Assume we have m flash memory cells indexed 0 to $m-1$, and with charge levels $c_0, \ldots, c_{m-1}$. We shall say the counter encodes the integer $z \in \mathbb{Z}$, $z \geq 0$, if $0 \leq z \leq m-2$ is the smallest non-negative integer for which $c_z > c_{z+1}$. If no such z exists, we shall say the counter encodes the value $m-1$. Using m cells the counter can assume the value of any integer in $\{0, \ldots, m-1\}$. We note that when $1 \leq s < t$, in the (s,t,n)-LRM scheme every cell is comparable with its predecessor and successor. Thus, if the counter represents the value z, increasing it to $z+1$ involves a single push-to-the-top operation on cell $z+1$, ensuring both $c_z < c_{z+1}$ and $c_{z+1} > c_{z+2}$ (or just the former, if $z+1 = m-1$). Resetting the counter to represent a 0 is equally simple, and requires a single push-to-the-top operation on cell 0, ensuring $c_0 > c_1$.

Construction 6.3: *The construction is a variation on Construction* 6.2*, so we shall describe it by noting the differences between them.*

We set $n = m^2 + m$, and each codeword shall be made up of $m+1$ blocks of length m. The first m blocks are those described in Construction 6.2*. Block m, the last block, will implement a counter. Thus, if g is a length m^2 codeword from Construction* 6.2*, and the counter represents the value z, we shall denote the codeword in the code we now construct as the pair (g, z).*

Let $g_0, g_1, \ldots, g_{L-1}$ be the anchor vectors from Construction 6.2*, and assume ℓ_i is the number of auxiliary vectors between $g_{\ell-1}$ and g_ℓ (indices taken modulo L) in Construction* 6.2*. The new code we construct has anchors*

$$g'_i = (g_i, m - \ell_i),$$

for all $i \in [L]$.

Finally, we create auxiliary vectors between adjacent anchors using Algorithm 6.2*, which is a simple variation on the original Algorithm* 6.1*.*

Intuitively, Algorithm 6.2 is the same as Algorithm 6.1 except for the counter reset at

Algorithm 6.2 Transform block k of configuration $\mathbf{f_c}$ from v_{s_i} to $v_{s_{i+m}}$ in a fixed number of steps

Input: current cell configuration $\mathbf{f_c}$, block number k, new block value $v_{s_{i+m}}$
Output: new cell configuration $\mathbf{f_{c'}}$
Reset the counter.
Push cell $m-1$ (the last cell) in block $k-1$ (modulo m).
$a_j \Leftarrow 0$ for all $j = 0, 1, \ldots, m-3$
$z \Leftarrow 0$
$j \Leftarrow 0$
repeat
  if $v_{s_{i+m}}(j) = \sum_{i=j+1}^{r'(j)} a_i$ and $a_j = 0$ **then**
    if $c_{km+j} \leq \max_{l'(j) \leq j' \leq r'(j), a_{j'}=1} c_{km+j'}$ **then**
      Push the jth cell in block k (the cell in position $km+j$).
      $z \Leftarrow z+1$
    end if
    $a_j \Leftarrow 1$
    $j \Leftarrow l'(j)$
  else
    $j \Leftarrow j+1$
  end if
until $j = m-2$
repeat
  Increment the counter.
  $z \Leftarrow z+1$
until $z = m$
Push cell $m-2$ (the next-to-last cell) in block k.
Output the resulting cell configuration $\mathbf{f_{c'}}$.

its beginning, and the counter increments at its end. These ensure the number of auxiliary vectors between anchors is constant. We observe the simple fact that anchor codewords have a non-zero counter, and so, when we reset the counter at the beginning of Algorithm 6.2 we obtain a different vector. Showing the codewords are distinct follows the exact same arguments as those used for Construction 6.2. In contrast with Construction 6.2, in Construction 6.3 we can give an exact expression for the size of the code, $L(m+2)$, since we have L anchors, and the distance between anchors is exactly $m+2$. It is also easy to show

that Corollary 6.2 also holds for a family of codes generated using Construction 6.3.

Finally, the next-state, encoding, and, decoding algorithms may all be implemented efficiently for the codes from Construction 6.3. The next-state algorithm is essentially the same as that for the codes from Construction 6.2. As for encoding and decoding, the ith codeword may be uniquely described using $i_{\text{anchor}} = \lfloor i/(m+2) \rfloor$ which is the index of the nearest previous anchor in the code, and by $i_{\text{aux}} = i \bmod (m+2)$ which is the distance from the previous anchor.

Decoding is done by identifying the underlined block, and thus retrieving the correct position in the De-Bruijn sequence (see [61, 84] for example), which in turn gives us i_{anchor}. If the codeword is not an anchor, we can run Algorithm 6.2 on the anchor until we reach the current codeword, and thus obtain i_{aux}. The decoded index is therefore $(m+2)i_{\text{anchor}} + i_{\text{aux}}$. Encoding is done in reverse, where we use an algorithm for encoding De-Bruijn sequences to find the appropriate De-Bruijn sub-sequence, translate it to the anchor of index i_{anchor}, and then run Algorithm 6.2 until i_{aux} push-to-the-top operations are made.

6.4 Summary

We presented the general framework of (s, t, n)-local rank modulation and focused on the specific case of $(1, 2, n)$-LRM, which is both the least-hardware-intensive and the simplest one to translate between binary strings and permutations. We studied constant-weight Gray codes for this scheme, which guarantee a bounded charge difference in any "push-to-the-top" operation. The Gray codes are used to simulate a conventional multi-level flash cell.

We started with a construction, where by letting w be approximately $n/2$ we obtained cyclic $(1, 2, n; w)$-LRMGCs whose rate approaches 1. We then studied Gray codes for the more general (s, t, n)-LRM. The codes we presented are asymptotically rate-optimal.

Several questions remain open. Constant-weight codes for the general (s, t, n)-LRM case are still missing. Of more general interest is the study of codes that cover a constant fraction of the space.

Bibliography

[1] E. Arıkan. Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Trans. on Inform. Theory*, 55(7):3051–3073, July 2009.

[2] E. Arikan. Source polarization. In *IEEE Inter. Symp. Infor,. Theory (ISIT)*, pages 899–903, 2010.

[3] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[4] E. F. Beckenbach (Editor). *Applied Combinatorial Mathematics*. New York, J. Wiley, 1964.

[5] C. H. Bennett, G. Brassard, and J-M Robert. Privacy amplification by public discussion. *SIAM J. Comput.*, 17(2):210–229, April 1988.

[6] M. Buck and D. Wiedemann. Gray codes with restricted density. *Discrete Math.*, 48:163–181, 1984.

[7] D. Burshtein and A. Strugatski. Polar write once memory codes. *IEEE Trans. on Inform. Theory*, 59(8):5088–5101, August 2013.

[8] P. Cappelletti, C. Golla, P. Olivo, and E. Zanoni. *Flash Memories*. Kluwer Academic Publishers, 1999.

[9] M. Carkeet and P. Eades. A subset generation algorithm with a very strong minimal change property. *Congressus Numerantium*, 47:139–143, 1985.

[10] V. Chandar, E. Martinian, and G. W. Wornell. Information embedding codes on graphs with iterative encoding and decoding. In *IEEE Inter. Symp. Infor,. Theory (ISIT)*, pages 866–870, July 2006.

[11] C. C. Chang, H. Y. Chen, and C. Y. Chen. Symbolic Gray code as a data allocation scheme for two-disc systems. *Comput. J.*, 35:299–305, 1992.

[12] M. Chen and K. G. Shin. Subcube allocation and task migration in hypercube machines. *IEEE Trans. on Comput.*, 39:1146–1155, 1990.

[13] T. M. Cover and J. A. Thomas. *Elements of information theory (2. ed.)*. Wiley, 2006.

[14] P. Diaconis and S. Holmes. Gray codes for randomization procedures. *Stat. Comput.*, 4:287–302, 1994.

[15] P. Eades, M. Hickey, and R. C. Read. Some Hamiltonian paths and minimal change algorithms. *J. of the ACM*, 31:19–29, 1984.

[16] A. El Gamal and Y.-H. Kim. *Network information theory*. Cambridge University Press, 2012.

[17] E. En Gad, A. Jiang, and J. Bruck. Compressed encoding for rank modulation. In *IEEE Inter. Symp. Infor,. Theory (ISIT)*, pages 884–888, July 2011.

[18] E. En Gad, A. Jiang, and J. Bruck. Trade-offs between instantaneous and total capacity in multi-cell flash memories. In *IEEE Inter. Symp. Infor,. Theory (ISIT)*, pages 990–994, July 2012.

[19] E. En Gad, M. Langberg, M. Schwartz, and J. Bruck. Constant-weight Gray codes for local rank modulation. *IEEE Trans. on Inform. Theory*, 57(11):7431–7442, November 2011.

[20] E. En Gad, M. Langberg, M. Schwartz, and J. Bruck. Generalized gray codes for local rank modulation. *IEEE Trans. on Inform. Theory*, 59(10):6664–6673, Oct 2013.

[21] E. En Gad, Y. Li, J. Kliewer, M. Langberg, A. Jiang, and J. Bruck. Polar coding for noisy write-once memories. In *IEEE Inter. Symp. Infor,. Theory (ISIT)*, pages 1638–1642, June 2014.

[22] E. En Gad, E. Yaakobi, A. Jiang, and J. Bruck. Rank-modulation rewriting codes for flash memories. In *IEEE Inter. Symp. Infor,. Theory (ISIT)*, pages 704–708, July 2013.

[23] C. Faloutsos. Gray codes for partial match and range queries. *IEEE Trans. on Software Eng.*, 14:1381–1393, 1988.

[24] H. C. Ferreira, A. J. Han Vinck, T. G. Swart, and I. de Beer. Permutation trellis codes. *IEEE Trans. on Communications*, pages 1782–1789, November 2005.

[25] A. Gabizon and R. Shaltiel. Invertible zero-error dispersers and defective memory with stuck-at errors. In *APPROX-RANDOM*, pages 553–564, 2012.

[26] R. G. Gallager. *Information Theory and Reliable Communication*. Wiley, 1968.

[27] M. Gardner. The curious properties of the Gray code and how it can be used to solve puzzles. *Scientif. Amer.*, 227:106–109, 1972.

[28] S.I. Gel'fand and M.S. Pinsker. Coding for channel with random parameters. *Problems of Control Theory*, 9(1):19–31, 1980.

[29] N. Goela, E. Abbe, and M. Gastpar. Polar codes for broadcast channels. *IEEE Trans. on Inform. Theory*, 61(2):758–782, Feb 2015.

[30] D. Goldin and D. Burshtein. Improved bounds on the finite length scaling of polar codes. *IEEE Trans. on Inform. Theory*, 60(11):6966–6978, Nov 2014.

[31] S. W. Golomb. *Shift Register Sequences*. Holden-Day, San Francisco, 1967.

[32] S.W. Golomb. The limiting behavior of the z-channel (corresp.). *IEEE Trans. on Inf. Theor.*, 26(3):372–372, May 1980.

[33] F. Gray. Pulse code communication, March 1953. U.S. Patent 2632058.

[34] M. Hagiwara and H. Imai. Quantum quasi-cyclic ldpc codes. In *IEEE Inter. Symp. Infor,. Theory (ISIT)*, pages 806–810, June 2007.

[35] M. Hamada. Conjugate codes for secure and reliable information transmission. In *IEEE Infor. Theor. Works. (ITW)*, pages 149–153, Oct 2006.

[36] C. Heegard. On the capacity of permanent memory. *IEEE Trans. on Inform. Theory*, 31(1):34–42, January 1985.

[37] J. Honda and H. Yamamoto. Polar coding without alphabet extension for asymmetric models. *IEEE Trans. on Inform. Theory*, 59(12):7829–7838, 2013.

[38] T. Hough and F. Ruskey. An efficient implementation of the Eades, Hickey, Read adjacent interchange combination generation algorithm. *J. of Comb. Math. and Comb. Comp.*, 4:79–86, 1988.

[39] R. Impagliazzo, L. A. Levin, and M. Luby. Pseudo-random generation from one-way functions. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, STOC '89, pages 12–24, New York, NY, USA, 1989. ACM.

[40] L. Ioffe and M. Mézard. Asymmetric quantum error-correcting codes. *Phys. Rev. A*, 75:032345, Mar 2007.

[41] A.N. Jacobvitz, R. Calderbank, and D.J. Sorin. Writing cosets of a convolutional code to increase the lifetime of flash memory. In *Allerton Conf. Comm., Contol, and Comp.*, pages 308–318, Oct 2012.

[42] A. Jiang, V. Bohossian, and J. Bruck. Rewriting codes for joint information storage in flash memories. *IEEE Trans. on Inform. Theory*, 56(10):5300–5313, 2010.

[43] A. Jiang, M. Langberg, M. Schwartz, and J. Bruck. Trajectory codes for flash memory. *IEEE Trans. on Inform. Theory*, 59(7):4530–4541, July 2013.

[44] A. Jiang, A. Li, E. En Gad, M. Langberg, and J. Bruck. Joint rewriting and error correction in write-once memories. In *IEEE Inter. Symp. Infor,. Theory (ISIT)*, pages 1067–1071, 2013.

[45] A¿ Jiang, R. Mateescu, M. Schwartz, and J. Bruck. Rank modulation for flash memories. *IEEE Trans. on Inform. Theory*, 55(6):2659–2673, June 2009.

[46] A. Jiang, M. Schwartz, and J. Bruck. Correcting charge-constrained errors in the rank-modulation scheme. *IEEE Trans. on Inform. Theory*, 56(5):2112–2120, May 2010.

[47] M. Kim, J.K. Park, and C.M. Twigg. Rank modulation hardware for flash memories. In *IEEE Int. Midwest Symp. on Circuits and Systems (MWSCAS)*, pages 294 –297, aug. 2012.

[48] Y. Kim, R. Mateescu, S-H Song, Z. Bandic, and B.V.K. Vijaya Kumar. Coding scheme for 3D vertical flash memory. *Available at http://arxiv.org/abs/1410.8541v2*, February 2015.

[49] Y. Kim and B.V.K. Vijaya Kumar. Writing on dirty flash memory. In *Allerton Conf. Comm., Contol, and Comp.*, pages 513–520, Sept 2014.

[50] S.B. Korada and R.L. Urbanke. Polar codes are optimal for lossy source coding. *IEEE Trans. on Inform. Theory*, 56(4):1751–1768, April 2010.

[51] A.V. Kusnetsov and B. S. Tsybakov. Coding in a memory with defective cells,. *translated from Problemy Peredachi Informatsii*, 10(2):52–60, 1974.

[52] C. A. Laisant. Sur la numération factorielle, application aux permutations. *Bulletin de la Société Mathématique de France*, 16:176–183, 1888.

[53] R. M. Losee. A Gray code based ordering for documents on shelves: classification for browsing and retrieval. *J. Amer. Soc. Inform. Sci.*, 43:312–322, 1992.

[54] J. M. Ludman. Gray codes generation for MPSK signals. *IEEE Trans. on Communications*, COM-29:1519–1522, 1981.

[55] Xudong Ma. Write-once-memory codes by source polarization. *Available at http://arxiv.org/abs/1405.6262*, May 2014.

[56] D.J.C. MacKay. Good error-correcting codes based on very sparse matrices. *IEEE Trans. on Inform. Theory*, 45(2):399–431, Mar 1999.

[57] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland, 1978.

[58] E. Martinian and J.S. Yedidia. Iterative quantization using codes on graphs. In *Allerton Conf. Comm., Contol, and Comp.*, 2003.

[59] K. Marton. A coding theorem for the discrete memoryless broadcast channel. *IEEE Trans. on Inf. Theor.*, 25(3):306–311, May 1979.

[60] O. Milenkovic and B. Vasic. Permutation (d,k) codes: efficient enumerative coding and phrase length distribution shaping. *IEEE Trans. on Inform. Theory*, 46(7):2671–2675, July 2000.

[61] C. J. Mitchell, T. Etzion, and K. G. Paterson. A method for constructing decodable de Bruijn sequences. *IEEE Trans. on Inform. Theory*, IT-42:1472–1478, 1996.

[62] M. Mondelli, I. Hassani, and R. Urbanke. How to achieve the capacity of asymmetric channels. *Available at http://arxiv.org/abs/1406.7373*, June 2014.

[63] M. Mondelli, S.H. Hassani, I. Sason, and R.L. Urbanke. Achieving Marton's region for broadcast channels using polar codes. *IEEE Trans. on Inform. Theory*, 61(2):783–800, Feb 2015.

[64] R. Mori and T. Tanaka. Channel polarization on q-ary discrete memoryless channels by arbitrary kernels. In *IEEE Inter. Symp. Infor,. Theory (ISIT)*, pages 894–898, June 2010.

[65] P. Oswald and A. Shokrollahi. Capacity-achieving sequences for the erasure channel. *IEEE Trans. on Inform. Theory*, 48(12):3017–3028, Dec 2002.

[66] W. Park and A. Barg. Polar codes for q-ary channels, $q = 2^r$. *IEEE Trans. on Inform. Theory*, 59(2):955–969, Feb 2013.

[67] T.V. Ramabadran. A coding scheme for m-out-of-n codes. *IEEE Trans. on Communications*, 38(8):1156–1163, August 1990.

[68] D. Richards. Data compression and Gray-code sorting. *Information Processing Letters*, 22:201–205, 1986.

[69] R.L. Rivest and A. Shamir. How to reuse a write-once memory. *Information and Control*, 55(1-3):1–19, 1982.

[70] J. Robinson and M. Cohn. Counting seqeuences. *IEEE Trans. on Comput.*, C-30:17–23, May 1981.

[71] F. Ruskey. Adjacent interchange generation of combinations. *J. of Algorithms*, 9:162–180, 1988.

[72] A.G. Sahebi and S.S. Pradhan. Multilevel polarization of polar codes over arbitrary discrete memoryless channels. In *Allerton Conf. Comm., Contol, and Comp.*, pages 1718–1725, Sept 2011.

[73] P.K. Sarvepalli, A. Klappenecker, and M. Rötteler. Asymmetric quantum codes: constructions, bounds and performance. *Proc. Roy. Soc. of Lond. A Mat.*, 465(2105):1645–1672, 2009.

[74] E. Sasoglu. Polar codes for discrete alphabets. In *IEEE Inter. Symp. Infor,. Theory (ISIT)*, pages 2137–2141, July 2012.

[75] E. Sasoglu. Polarization and polar codes. *Foundations and Trends in Communications and Information Theory*, 8(4):259–381, 2012.

[76] E. Sasoglu, I.E. Telatar, and E. Arikan. Polarization for arbitrary discrete memoryless channels. In *IEEE Infor. Theor. Works. (ITW)*, pages 144–148, Oct 2009.

[77] C. D. Savage. A survey of combinatorial Gray codes. *SIAM Rev.*, 39(4):605–629, December 1997.

[78] R. Sedgewick. Permutation generation methods. *Computing Surveys*, 9(2):137–164, June 1977.

[79] A. Shpilka. Capacity-achieving multiwrite wom codes. *IEEE Trans. on Inform. Theory*, 60(3):1481–1487, March 2014.

[80] D. Sutter, J.M. Renes, F. Dupuis, and R. Renner. Achieving the capacity of any dmc using only polar codes. In *IEEE Infor. Theor. Works. (ITW)*, pages 114–118, Sept 2012.

[81] I Tal and A Vardy. How to construct polar codes. *IEEE Trans. on Inf. Theor.*, 59(10):6562–6582, Oct 2013.

[82] I. Tamo and M. Schwartz. Correcting limited-magnitude errors in the rank-modulation scheme. *IEEE Trans. on Inform. Theory*, 56(6):2551–2560, June 2010.

[83] C. Tian, V.A. Vaishampayan, and N. Sloane. A coding algorithm for constant weight vectors: A geometric approach based on dissections. *IEEE Trans. on Inform. Theory*, 55(3):1051–1060, March 2009.

[84] J. Tuliani. De Bruijn sequences with efficient decoding algorithms. *Discrete Math.*, 226(1):313–336, January 2001.

[85] Z. Wang, A. Jiang, and J. Bruck. On the capacity of bounded rank modulation for flash memories. In *IEEE Inter. Symp. Infor,. Theory (ISIT)*, pages 1234–1238, June 2009.

[86] E. Yaakobi, S. Kayser, P.H. Siegel, A. Vardy, and J.K. Wolf. Codes for write-once memories. *IEEE Trans. on Inform. Theory*, 58(9):5985–5999, September 2012.

[87] E. Yaakobi, H. Mahdavifar, P.H. Siegel, A. Vardy, and J.K. Wolf. Rewriting codes for flash memories. *IEEE Trans. on Inform. Theory*, 60(2):964–975, Feb 2014.

[88] E. Yaakobi, P.H. Siegel, A. Vardy, and J.K. Wolf. Multiple error-correcting wom-codes. *IEEE Trans. on Inform. Theory*, 58(4):2220–2230, April 2012.

[89] Y. Yehezkeally and M. Schwartz. Snake-in-the-box codes for rank modulation. *IEEE Trans. on Inform. Theory*, 58(8):5471–5483, Aug 2012.

[90] R. Zamir, S. Shamai, and U. Erez. Nested linear/lattice codes for structured multiterminal binning. *IEEE Trans. on Inform. Theory*, 48(6):1250–1276, Jun 2002.

[91] G. Zemor and G. D. Cohen. Error-correcting wom-codes. *IEEE Trans. on Inform. Theory*, 37(3):730–734, May 1991.