UNIVERSITY OF CALIFORNIA, SAN DIEGO

**Constrained Codes and Signal Processing for Non-Volatile Memories**

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

in

Electrical Engineering
(Communication Theory and Systems)

by

Minghai Qin

Committee in charge:

Professor Paul H. Siegel, Chair
Professor Alon Orlitsky
Professor Bhaska Rao
Professor Steven Swanson
Professor Alexander Vardy

2014

The dissertation of Minghai Qin is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____
Chair

University of California, San Diego

2014

# DEDICATION

*Dedicated to my parents.*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

Many thanks to many people who contribute to my work in the past five years.

First, I would like to express my deepest gratitude for my advisor Prof. Paul H. Siegel. His rigorousness in research, sharpness in thought, and kindness in life have profoundly affected my attitude towards work and life. His guidance have simplified the world of coding and signal processing, which enables me to change from unaware to educated. It is a great pleasure to discuss with him and I would like to thank Prof. Siegel for his patience and belief in me that gives me the freedom to search for and explore my interests.

It is fortunate for me to study in UCSD where there are a plenty of outstanding researchers in the field of communication and signal processing. I would like to thank Prof. Alon Orlitsky for the introduction of information theory when I joined UCSD five year ago. His brain-storming teaching methods guided me to the world of entropy. I would like to thank Prof. Bhaskar Rao for his introductory course to time series analysis. I would like to thank Prof. Steven Swanson and Prof. Alexander Vardy, together with Prof. Alon Orlitsky, Prof. Bhaskar Rao, and Prof. Siegel, for their time and effort to serve in my committee. I would like to thank Prof. Young-Han Kim for the opportunity to collaborate with his group to look at problems from a different information theoretic angle. I would like to thank Prof. Eitan Yaakobi in Technion - Israel Institute of Technology for many collaborations on coding and signal processing on flash memories. I would like to thank Prof. Anxiao (Andrew) Jiang in Texas A&M University for the collaboration during his visit to UCSD and thank Prof. Albert Guillén i Fàbregas in Universitat Pompeu Fabra (UPF) for the collaboration with his group and inviting me to visit UPF. Finally, I would like to thank Prof. Jack K. Wolf and I cherish my memories of sitting in his classroom for my first graduate course. He was a wise man. He led me to this field.

I would like to thank all my collaborators, Rajiv Agarwal, Aman Bhatia, John Cioffi, Albert Guillén i Fàbregas, Jing Guo, Aravind Iyengar, Anxiao Jiang, Young-Han Kim, Brian Kurkoski, Borja Peleato, Lele Wang , and Eitan Yaakobi, for their great contributions to my dissertation.

I would like to thank CMRR staff, Ray Descoteaux, Betty Manoulian, Julie Matsuda, and Iris Villanueva for their kind help in making my life in CMRR more enjoyable and easier.

I would like to thank the current members and alumni of STAR group: Aman Bhatia, Brian Butler, Bing Fan, Pengfei Huang, Aravind Iyengar, Seyhan Karakulak, Scott Kayser, Ido Tal, Veeresh Taranalli, Han Wang, Eitan Yaakobi, and Xiaojie Zhang. I have benefited a lot during the discussion and conversation with them.

A special thanks to my parents, Ms. Yongchen Zhu and Mr. Rong Qin. You gave me birth and wisdom. I appreciate their considerate love and unconditional support.

I would like to thank my friends in San Diego. Zheng Li, Sibo Tao, Lele Wang, Shaohe Wang, Lelin Zhang, Jun Zhou, etc. Their friendship enriches my ex-curriculum life and for the past five years they have always been there, ready for help.

This thesis was supported in part by the ISEF Foundation, the Lester Deutsch Fellowship, the University of California Lab Fees Research Program, Award No. 09-LR-06-118620-SIEP, the Qualcomm Innovation Fellowship, the NSF under Grant CCF-1116739, and the Center for Magnetic Recording Research at the University of California, San Diego.

Chapter 2 is in part a reprint of the material in the paper: Minghai Qin, Eitan Yaakobi, and Paul H. Siegel, "Time-space constrained codes for phase-change memories", *IEEE Transaction on Information Theory*, vol. 59, no. 8, pp. 5102-5114, August 2013.

Chapter 3 is in part a reprint of the material in the paper: Aman Bhatia, Minghai Qin, Aravind Iyengar, Brian Kurkoski, and Paul H. Siegel, "Lattice-based WOM codes for multilevel flash memories", *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 933-945, May 2014.

Chapter 4 is in part a reprint of the material in the paper: Lele Wang, Minghai Qin, Eitan Yaakobi, Young-Han Kim, and Paul H. Siegel, "WOM with retained messages", in *Proc. IEEE International Symposium of Information Theory (ISIT) 2012*, Cambridge, MA, USA, July 2012, pp. 1396-1400.

Chapter 5 is in part a reprint of the material in the paper: Minghai Qin, Eitan Yaakobi, and Paul H. Siegel, "Constrained codes that mitigate intercell interference in read/write cycles for flash memories", *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 836-846, May 2014.

Chapter 6 is in part a reprint of the material in the paper: Minghai Qin, Eitan Yaakobi, and Paul H. Siegel, "Optimized cell programming for flash memories with quantizers", *IEEE Transaction on Information Theory*, vol. 60, no. 5, pp. 1-16, May 2014.

Chapter 7 is in part a reprint of the material in the paper: Minghai Qin, Anxiao Jiang, and Paul H. Siegel, "Parallel programming of rank modulation", in *Proc. IEEE International Symposium of Information Theory (ISIT) 2013*, Istanbul, Turkey, July 2013, pp. 719-723.

## VITA

| | |
|---|---|
| 2009 | Bachelor of Engineering in Electronic Engineering, Tsinghua University |
| 2011 | Master of Science in Electrical and Computer Engineering, University of California, San Diego |
| 2014 | Doctor of Philosophy in Electrical Engineering, University of California, San Diego |

## PUBLICATIONS

Minghai Qin, Eitan Yaakobi, and Paul H. Siegel, "Optimized cell programming for flash memories with quantizers", *IEEE Transaction on Information Theory*, vol. 60, no. 5, pp. 1-16, May 2014

Minghai Qin, Eitan Yaakobi, and Paul H. Siegel, "Constrained codes that mitigate intercell interference in read/write cycles for flash memories", *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 836-846, May 2014

Aman Bhatia, Minghai Qin, Aravind Iyengar, Brian Kurkoski, and Paul H. Siegel, "Lattice-based WOM codes for multilevel flash memories", *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 933-945, May 2014

Minghai Qin, Eitan Yaakobi, and Paul H. Siegel, "Time-space constrained codes for phase-change memories", *IEEE Transaction on Information Theory*, vol. 59, no. 8, pp. 5102-5114, August 2013

Jing Guo, Minghai Qin, Albert Guillén i Fàbregas, and Paul H. Siegel, "Enhanced belief propagation decoding of polar codes through concatenation", in *Proc. IEEE International Symposium of Information Theory (ISIT) 2014*, Honolulu, HI, USA, July 2014

Minghai Qin, Anxiao Jiang, and Paul H. Siegel, "Parallel programming of rank modulation", in *Proc. IEEE International Symposium of Information Theory (ISIT) 2013*, Istanbul, Turkey, July 2013, pp. 719-723

Borja Peleato, Rajiv Agarwal, John Cioffi, Minghai Qin and Paul H. Siegel, "Towards minimizing read time for NAND Flash", in *Proc. IEEE Globecom 2012*, Anaheim, CA, USA, December 2012, pp. 3219-3224

Minghai Qin, Eitan Yaakobi, and Paul H. Siegel, "Optimized cell programming for flash memories with quantizers", in *Proc. IEEE International Symposium of Information Theory (ISIT) 2012*, Cambridge, MA, USA, July 2012, pp. 995-999

Lele Wang, Minghai Qin, Eitan Yaakobi, Young-Han Kim, and Paul H. Siegel, "WOM with retained messages", in *Proc. IEEE International Symposium of Information Theory (ISIT) 2012*, Cambridge, MA, USA, July 2012, pp. 1396-1400

Minghai Qin, Eitan Yaakobi, and Paul H. Siegel, "Time-space constrained codes for phase-change memories", in *Proc. IEEE Globecom 2011*, Houston, Texas, USA, December 2011, pp. 1-6

ABSTRACT OF THE DISSERTATION

**Constrained Codes and Signal Processing for Non-Volatile Memories**

by

Minghai Qin

Doctor of Philosophy in Electrical Engineering
(Communication Theory and Systems)

University of California, San Diego, 2014

Professor Paul H. Siegel, Chair

Non-volatile memories (NVMs) have attracted considerable attention as data storage media because of their fast read/write speed, high data throughput, large capacity, and low power consumption. They are widely used in mobile, embedded, and mass-storage applications. The research goal of this dissertation is to model NVM systems in a mathematical manner and to design constrained codes and signal processing techniques that improve the performance of NVM systems.

Our main contribution of this research dissertation is the invention of constrained codes for different NVM systems. For the most widely-used flash memory systems, we manage to use inter-cell interference (ICI) free codes to mitigate the coupling effect of adjacent cells during write operations, to use balanced codes to tolerate charge leakage effects during read operations, and to use write-once memory (WOM) codes to enlarge the lifetime capacity of devices for sev-

eral practical scenarios. On the other hand, we also manage to use time-space constrained codes to eliminate cross-talk and to reduce heat accumulation for phase-change memories (PCMs), which is one of the emerging technologies that could potentially replace flash memories as the building block for the next generation of NVM systems. We derive the capacity (or bounds on the capacity) for each constraint and provide explicit code constructions with low encoding and decoding complexities.

Furthermore, the optimization of programming flash memory systems is studied by signal processing tools. Flash memories use the amount of charge trapped in cells to represent the data and the cells can have multiple levels . One of the most prominent features of programming flash memory cells is the asymmetry in programming and erasing, namely, the cell levels can only be increased during programming and cell erasure must be done at the block level. In order to achieve high programming speed and precision, we solve optimization problems that either minimize the programming time or the number of programming errors, subject to the constraint of unidirectional incrementing of cell levels.

# Chapter 1

# Introduction

## 1.1 Background

Non-volatile memory (NVM) is computer memory that can preserve stored information even when not powered. It includes read-only memories (ROMs), (e.g., PROM, EPROM, EEPROM), non-volatile random-access memories (RAMs), (e.g., flash memories, phase-change memories), etc. The non-volatile property enables NVMs to be used in a wide range of data storage applications, such as cellphones, cameras, computers, and so on. Due to their low power consumption, large read/write throughput, and high density, there is a growing interest in research on advanced NVMs, especially in the most widely-used flash memories and new technologies including the phase-change memory (PCM). In particular, this dissertation focuses on the research problems that arise from signal processing and coding algorithms that enhance the capacities of flash memories and PCMs.

The challenges of implementing NVMs come from the need for improved data reliability. Typically, NVM systems consist of arrays of cells and each cell has multiple states to represent data. During the writing or programming process, the data intended to be written to the memories might be distorted due to programming noise, write disturbance, and cell heterogeneity; during the reading process, the data read from the NVM systems might also be corrupted due to read noise, read disturbance, and aging of the NVM cells. All of the mechanisms above would cause asymmetric, time-variant, and unpredictable errors in NVM systems. Furthermore, as the NVM cells are arranged in a denser array to allow higher capacity per square inch, the interference between NVM cells is no longer negligible and more attention needs to be paid to cross-talk of adjacent cells.

Coding and signal processing tools play an important role in detecting and eliminating the errors in NVM systems. The data storage system can be viewed as a communication system through the dimension of time, where one wishes to communicate from the present to the future. Shannon established the theory of information and coding in his celebrated paper [75] and determined the maximum achievable transmission rate of a communication system. Since then, a variety of codes have been discovered and implemented in data storage systems. These codes include error correction codes (ECCs) and constrained codes. ECCs use extra redundancy to correct the corrupted data in storage devices. Examples of ECCs include Hamming codes, Reed-Solomon (RS) codes, BCH codes, Turbo codes, low-density parity-check (LDPC) codes, and recently discovered polar codes and spatially coupled codes. On the other hand, constrained codes forbid some error-prone patterns from being recorded in the storage system. For example, run-length limited (RLL) constraints [96] and maximum transition run (MTR) constraints [64] are used in magnetic recording to avoid inter-symbol interference and to avoid synchronization errors. Additionally, signal processing techniques provide a powerful tool for data detection and estimation.

## 1.2 Constrained codes for non-volatile memories

Constrained codes are capable of mitigating the side effects of write asymmetry and ICI by not allowing the cell state to change arbitrarily and removing the most error-prone patterns. Applications of constrained codes date back to tape drives and hard-disk drives (HDDs). For example, for binary sequences, the $(d, k)$ run-length limited (RLL) constraint [96] requires the runs of 0's between successive 1's to be greater than or equal to $d$ (the $d$-constraint) and less than or equal to $k$ (the $k$-constraint). The $d$-constraint reduces the effect of inter-symbol interference and the $k$-constraint aids in timing control. The capacity, which describes the asymptotic growth rate of the number of sequences satisfying a constraint, is one of the most important parameters related to the constraint. The study of the capacity of various constraints originated in 1948 from [75]. Since then, methods for the construction of constrained codes based on enumerative coding techniques and finite-state graphs have been developed.

NVM cells are arranged as 1-dimensional or 2-dimensional arrays. Typically, the detection/change of cell states (i.e. read/write process) is driven electronically. The increasing demands for larger capacities require the cells to be grouped into denser arrays, resulting in the interference between neighboring cells during the writing process. The parasitic capacitance between adjacent cells in flash memory, as an example, causes inter-cell interference (ICI) such that

the level of one cell could increase unexpectedly when high voltage is applied to its neighboring cells. On the other hand, the intrinsic properties of NVM cells can also lead to read errors, such as charge leakage of flash memory cells, crystallization for PCM cells, and all types of degradation of NVM cells. The mechanisms can unpredictably change the cell states with the passing of time. In this dissertation, we study several constraints and the construction of representative constrained codes that can reduce read/write errors and extend the lifetime of NVMs.

## 1.3 Signal processing for non-volatile memories

The read/write process of NVM systems is restricted by intrinsic properties of NVM cells. In order to achieve a better precision, a higher speed, and a longer lifetime of NVM systems, the read/write process is carefully controlled. Signal processing techniques provide both criteria and guidelines in the optimization of the read/write process.

In this dissertation, we consider the write process of flash memories. Flash memory cells use floating-gate transistors to store data and the data is represented by a cell-state vector. The $q$-ary cell-state is usually determined by discretizing the amount of trapped charge (e.g. electrons) in a cell. The read and write process corresponds to the evaluation of the amount of electrons in flash memory cells and the injection of electrons to the flash memory cells. The unit of read/write operations in flash memory systems is a page, which consists of $\sim 10^3$ flash cells. One of the most prominent features in the writing process of flash memory systems is the asymmetry in programming and erasing. Programming (i.e., changing cell-states from low to high) can be accomplished by a single-page operation; however, erasing (i.e., changing cell-state from high to low) forces the whole block of cells ($\sim 10^6$ cells) to be erased. This operation, called block erasure, not only is time consuming but degrades the flash memory cells as well. Therefore, electrons are carefully injected when programming to avoid overshooting problems. In this dissertation, programming strategies that incur the fewest errors and lead to highest write speed are obtained by solving constrained optimization problems where the constraint function reflects the limitation of unidirectional incrementing of cell states during programming.

## 1.4 Dissertation Overview

In this dissertation, we first propose and study some constrained codes that fit into the models of different NVM systems. Then we study programming strategies for the flash memory system that acknowledge its unique asymmetry in programming and erasing.

In Chapter 2, we study time-space constrained codes for PCMs. In particular, this set of constrained codes can reduce the cross-talk induced by programming PCM cells and prolong the lifetime of PCM cells by limiting the heat generated during programming. Capacity analysis of the set of constraints is presented, followed by specific code constructions.

In Chapter 3, the write-once memory (WOM) code is introduced as an efficient rewriting code to increase the life-time capacity of flash memories. We study the construction of multi-level WOM codes based on integer lattices where we require that the data rates on all rewriting generations are equal. Asymptotically optimal code constructions are given for the case of $n = 2$ cells. For $n > 2$ cells, the asymptotically optimal partition for maximizing lifetime capacity of the $n$-dimensional integer lattice is presented. It remains open whether there exists a WOM code that is compatible with this partition.

In Chapter 4, we extend the WOM model such that during each rewriting process, not only new data is stored, but a subset of old data is required to be recoverable as well. Three canonical and concrete problems are proposed and the corresponding capacity analysis and code constructions are presented.

In Chapter 5, we study both the inter-cell interference (ICI) phenomenon in the write process and the charge leakage effect in the read process of flash memory systems. In the first part, constrained codes that mitigate the ICI and tolerate the charge leakage are proposed. Based on different techniques, asymptotic rate analysis for the constrained codes is presented. In the second part, a WOM model with mitigated ICI is explored. Both capacity analysis and code constructions are given.

In Chapter 6, we propose a more realistic criterion to evaluate the performance of programming techniques for flash memory cells and look for algorithms to optimize the parallel programming of an array of cells. In the first part, we present a polynomial time-complexity algorithm to achieve optimal programming in the absence of noise, either with or without ICI mitigation. In the second part, we study different models for programming noise and derive optimal programming strategies, with or without feedback information on cell levels after each round of programming.

In Chapter 7, we consider rank modulation, a data representation scheme that can tolerate overshooting in cell programming as well as charge leakage. We study parallel programming for rank modulation and minimization of programming time during data update. We present lower bounds on programming time based on both analytical equations and specific parallel programming algorithms and compare to the single-cell programming strategies.

# Chapter 2

# Time-space constrained codes for phase-change memories

## 2.1 Introduction

In Chapter 1, we briefly discussed that constrained codes can be used to improve the performance and lifetime of non-volatile memories. In this chapter, we focus on the application of constrained codes to phase-change memories.

Phase-change memory (PCM) devices are a promising technology for non-volatile memories. Like a flash memory, a PCM consists of cells that can be in distinct physical states. In the simplest case, the PCM cell has two possible states, an amorphous state and a crystalline state. Multiple-bit per cell PCMs can be implemented by using partially crystalline states [10].

While in a flash memory one can decrease a cell level only by erasing the entire block of about $10^6$ cells that contains it, in a PCM one can independently decrease an individual cell level – but only to level zero. This operation is called a RESET operation. A SET operation can then be used to change the cell state to any valid level. Therefore, in order to decrease a cell level from one non-zero value to a smaller non-zero value, one needs to first RESET the cell to level zero, and then SET it to the new desired level [10]. Thus, as with flash memory programming, there is a significant asymmetry between the two operations of increasing and decreasing a cell level.

As in a flash memory, a PCM cell has a limited lifetime; the cells can tolerate only about $10^7 - 10^8$ RESET operations before beginning to degrade [26]. Therefore, it is still important when programming cells to minimize the number of RESET operations. Furthermore, a RESET

operation can negatively affect the performance of a PCM in other ways. One of them is due to the phenomenon of thermal crosstalk. When a cell is RESET, the levels of its adjacent cells may inadvertently be increased due to heat diffusion associated with the operation [10, 68]. Another problem, called thermal accumulation, arises when a small area is subjected to a large number of program operations over a short period of time [10, 68]. The resulting accumulation of heat can significantly limit the minimum write latency of a PCM, since the programming accuracy is sensitive to temperature. It is therefore desirable to balance the thermal accumulation over a local area of PCM cells in a fixed period of time. Coding schemes can help overcome the performance degradation resulting from these physical phenomena. Lastras et al. [56] studied the capacity of a Write-Efficient Memory (WEM) [2] for a cost function that is associated with the write model of phase-change memories described above.

Jiang et al. [44] have proposed codes to mitigate thermal cross-talk and heat accumulation effects in PCM. Under their thermal cross-talk model, when a cell is RESET, the levels of its immediately adjacent cells may also be increased. Hence, if these neighboring cells exceed their target level, they also will have to be RESET, and this effect can then propagate to many more cells. In [44], they considered a special case of this and proposed the use of constrained codes to limit the propagation effect. Capacity calculations for these codes were also presented.

The other problem addressed in [44] is that of heat accumulation. In this model, the *rewrite cost* is defined to be the number of programmed cells, i.e., the Hamming distance between the current and next cell-state vectors. A code is said to be $(\alpha, \beta, p)$*-constrained* if for any $\alpha$ consecutive rewrites and for any segment of $\beta$ contiguous cells, the total rewrite cost of the $\beta$ cells over those $\alpha$ rewrites is at most $p$. A specific code construction was given for the $(\alpha \geqslant 1, \beta = 1, p = 1)$-constraint as well as an upper bound on the achievable rate of codes for this constraint. An upper bound on the achievable rate was also given for $(\alpha = 1, \beta \geqslant 1, p = 1)$-constrained codes.

The work in [44] dealt with only a few instances of the parameters $\alpha, \beta$, and $p$. In this chapter, we extend the code constructions and achievable-rate bounds to a larger portion of the parameter space. In Section 2.2, we formally define the constrained-coding problem for PCM. In Section 2.3, using connections to two-dimensional constrained coding, we present a scheme to calculate an upper bound on the achievable rate for all values of $\alpha, \beta$, and $p$. If the value of $\alpha$ or $\beta$ is 1 then the two-dimensional constraint becomes a one-dimensional constraint and we calculate the upper bound on the achievable rate for all values of $p$. This result coincides with the result in [44] for $(\alpha \geqslant 1, \beta = 1, p = 1)$ and $(\alpha = 1, \beta \geqslant 1, p = 1)$. We also derive upper bounds for some cases with parameters satisfying $(\alpha > 1, \beta > 1, p = 1)$ using known results

on the upper bound of the rate of two-dimensional constrained codes. In Section 2.4, several code constructions are given. First, we describe an elementary construction for arbitrary values of $\alpha, \beta$, and $p$. We then show an improved construction for $(\alpha = 1, \beta \geqslant 1, p \geqslant 1)$-constrained codes and extend the construction in [44] of $(\alpha \geqslant 1, \beta = 1, p = 1)$-constrained codes to arbitrary $p$. Finally, we show how to extend the improved constructions to arbitrary values of $\alpha, \beta$, and $p$.

## 2.2 Preliminaries

In this section, we give a formal definition of the constrained-coding problem. The number of cells is denoted by $n$ and the memory cells are binary. The cell-state vectors are the binary vectors from $\{0,1\}^n$. If a cell-state vector $\boldsymbol{u} = (u_1, \ldots, u_n) \in \{0,1\}^n$ is rewritten to another cell-state vector $\boldsymbol{v} = (v_1, \ldots, v_n) \in \{0,1\}^n$, then the rewrite cost is defined to be the Hamming distance between $\boldsymbol{u}$ and $\boldsymbol{v}$, that is

$$d_H(\boldsymbol{u}, \boldsymbol{v}) = |\{i \,:\, u_i \neq v_i, 1 \leqslant i \leqslant n\}|.$$

The Hamming weight of a vector $\boldsymbol{u}$ is $wt(\boldsymbol{u}) = d_H(\boldsymbol{u}, \boldsymbol{0})$. The complement of a vector $\boldsymbol{u}$ is $\overline{\boldsymbol{u}} = (\overline{u}_1, \ldots, \overline{u}_n)$. For a vector $\boldsymbol{x} = (x_1, \ldots, x_n)$, we denote by $\boldsymbol{x}_p^q$ the subvector $(x_p, x_{p+1}, \ldots, x_q)$ and for a sequence of vectors $\boldsymbol{x}_i = (x_{i,1}, \ldots, x_{i,n}), i \in \mathbb{N}$, we denote by $\boldsymbol{x}_{i,p}^q$ the subvector $(x_{i,p}, x_{i,p+1}, \ldots, x_{i,q})$, for $1 \leqslant p \leqslant q \leqslant n$. The set $\{i, i+1, \ldots, j\}$ is denoted by $[i : j]$ for $i \leqslant j$, and in particular, $\{1, 2, \ldots, \lfloor 2^{nR} \rfloor\}$ is denoted by $[1 : 2^{nR}]$ for an integer $n$ and real $R$.

We will specify a code by an explicit construction of its encoding and decoding maps. On the $i$-th write, for $i \geqslant 1$, the encoder

$$\mathcal{E}_i : [1 : 2^{nR_i}] \times \{0,1\}^n \mapsto \{0,1\}^n$$

maps the new information symbol and the current cell-state vector to the next cell-state vector. The decoder

$$\mathcal{D}_i : \{0,1\}^n \mapsto [1 : 2^{nR_i}]$$

maps the cell-state vector to the represented information symbol. We denote the *individual rate* on the $i$-th write of a code by $R_i$. Note that the alphabet size of the messages on each write does not have to be the same. The *rate* $R$ of the a code is defined as

$$R = \liminf_{m \to \infty} \frac{\sum_{i=1}^{m} R_i}{m}. \tag{2.1}$$

**Remark 2.2.1.** The limit $R$ exists: since the individual rates $R_i, i \in \mathbb{N}$, are bounded from above by 1, so are the average rates $\frac{\sum_{i=1}^{m} R_i}{m}$, for all $m \geqslant 1$.

**Definition 2.2.1.** *Let $\alpha, \beta, p$ be positive integers. A code $\mathcal{C}$ satisfies the* $(\alpha, \beta, p)$ ***time-space constraint*** *(or simply $(\alpha, \beta, p)$-constraint) if for any $\alpha$ consecutive rewrites and for any segment of $\beta$ contiguous positions, the total rewrite cost of those $\beta$ positions over those $\alpha$ rewrites is at most $p$. That is, if $\boldsymbol{v}_i = (v_{i,1}, \ldots, v_{i,n})$, for $i \geqslant 1$, is the cell-state vector on the $i$-th write, then, for all $i \geqslant 1$ and $1 \leqslant j \leqslant n - \beta + 1$,*

$$\left| \{(k, \ell) : v_{i+k,j+\ell} \neq v_{i+k+1,j+\ell}, 0 \leqslant k < \alpha, 0 \leqslant \ell < \beta\} \right| \leqslant p,$$

*or equivalently,*

$$\sum_{k=0}^{\alpha-1} d_H(\boldsymbol{v}_{i+k,j}^{j+\beta-1}, \boldsymbol{v}_{i+k+1,j}^{j+\beta-1}) \leqslant p.$$

*We call such a code $\mathcal{C}$ an* $(\alpha, \beta, p)$***-constrained code****.*

We assume that the number of writes is large and in the constructions we present there will be a periodic sequence of writes. Thus, it will be possible to change any $(\alpha, \beta, p)$-constrained code $\mathcal{C}$ with varying individual rates to an $(\alpha, \beta, p)$-constrained code $\mathcal{C}'$ with fixed individual rates such that the rates of the two constrained codes are the same. This can be achieved by using multiple copies of the code $\mathcal{C}$ and in each copy of $\mathcal{C}$ to start writing from a different write within the period of writes. Therefore, we assume that there is no distinction between the two cases and the rate is as defined in Equation (2.1), which is the average number of bits written per cell per write.

The encoding and decoding maps can be either the same on all writes or can vary among the writes. In the latter case, we will need more cells in order to record the index of the write number. However, arguing as in [95], it is possible to show that these extra cells do not reduce the asymptotic rate and therefore we assume here that the encoder and decoder know the write number.

A rate $R$ is called an $(\alpha, \beta, p)$***-achievable rate*** if there exists a sequence of $(\alpha, \beta, p)$-constrained codes of increasing length $n$ such that the rate of each code is $R$. The $(\alpha, \beta, p)$***-capacity*** of the $(\alpha, \beta, p)$-constraint is denoted by $C(\alpha, \beta, p)$ and is defined to be

$$C(\alpha, \beta, p) = \sup R,$$

where $R$ is an $(\alpha, \beta, p)$-achievable rate.

Our goal in this chapter is to give lower and upper bounds on the $(\alpha, \beta, p)$-capacity, $C(\alpha, \beta, p)$, for all values of $\alpha, \beta$, and $p$. Clearly, if $p \geqslant \alpha\beta$ then $C(\alpha, \beta, p) = 1$. So we assume

throughout the chapter that $p < \alpha\beta$. Lower bounds will be inferred from specific constrained code constructions while the upper bounds will be derived analytically using tools drawn from the theory of one- and two-dimensional constrained codes.

## 2.3 Upper Bound on the Capacity

In this section, we will present upper bounds on the $(\alpha, \beta, p)$-capacity obtained using techniques from the analysis of two-dimensional constrained codes. There are a number of two-dimensional constraints that have been extensively studied, e.g., 2-dimensional $(d, k)$-runlength-limited (RLL) constraints [50,82], the no isolated bits (n.i.b) constraint [23,34], and the family of checkerboard constraints [65, 90]. Given a two-dimensional constraint $S$, its capacity is defined to be

$$C_{2D}(S) = \lim_{m,n\to\infty} \frac{\log_2 c_S(m,n)}{mn},$$

where $c_S(m,n)$ is the number of $m \times n$ arrays that satisfy the constraint $S$. The constraint of interest for us in this work is the one where in each rectangle of size $a \times b$, the number of ones is at most $p$.

**Definition 2.3.1.** *Let $a, b, p$ be positive integers. An $(m \times n)$-array $A = (a_{i,j})_{1\leqslant i\leqslant m, 1\leqslant j\leqslant n} \in \{0,1\}^{m\times n}$ is called an $(a, b, p)$-*array *if in each sub-array of $A$ of size $a \times b$, the number of 1's is at most $p$. That is, for all $1 \leqslant i \leqslant m - a + 1, 1 \leqslant j \leqslant n - b + 1$,*

$$\left|\{(k,\ell) \,:\, 0 \leqslant k \leqslant a-1, 0 \leqslant \ell \leqslant b-1, a_{i+k,j+\ell} = 1\}\right| \leqslant p.$$

*The capacity of the constraint is denoted by $C_{2D}(a, b, p)$.*

Note that when $p = 1$, the $(a, a, 1)$-constraint coincides with the square checkerboard constraint of order $a - 1$ [90].

The connection between the capacity of the two-dimensional constraint $C_{2D}(a, b, p)$ and the $(\alpha, \beta, p)$-capacity is the following.

**Theorem 2.3.2.** *For all $\alpha, \beta, p$, $C(\alpha, \beta, p) \leqslant C_{2D}(\alpha, \beta, p)$.*

**Proof.** Let $\mathcal{C}$ be an $(\alpha, \beta, p)$-constrained code of length $n$. For any sequence of $m$ writes, let us denote by $\boldsymbol{v}_i$, for $i \geqslant 0$, the cell-state vector on the $i$-th write, where $\boldsymbol{v}_0$ is the all-zero vector. The $(m \times n)$-array $A = (a_{i,j})$ is defined to be

$$a_{i,j} = v_{i,j} + v_{i-1,j},$$

where the addition is a modulo-2 sum. That is, $a_{i,j} = 1$ if and only if the $j$-th cell is changed on the $i$-th write. Since $\mathcal{C}$ is an $(\alpha, \beta, p)$-constrained code, for all $1 \leqslant i \leqslant m - \alpha$ and $1 \leqslant j \leqslant n - \beta + 1$,

$$\left|\{(k, \ell) : v_{i+k,j+\ell} \neq v_{i+k+1,j+\ell}, 0 \leqslant k < \alpha, 0 \leqslant \ell < \beta\}\right| \leqslant p,$$

and therefore

$$\left|\{(k, \ell) \ : \ 0 \leqslant k \leqslant \alpha - 1, 0 \leqslant \ell \leqslant \beta - 1, a_{i+k,j+\ell} = 1\}\right| \leqslant p.$$

Thus, $A$ is an $(\alpha, \beta, p)$-array of size $m \times n$.

Every write sequence of the code $\mathcal{C}$ corresponds to an $(\alpha, \beta, p)$-array and thus the number of write sequences of length $m$ is at most the number of $(\alpha, \beta, p)$-arrays, which is upper bounded by $2^{mnC_{2D}(\alpha,\beta,p)}$, for $m, n$ large enough. Hence, the number of distinct write sequences is at most $2^{mnC_{2D}(\alpha,\beta,p)}$. However, if the individual rate on the $i$-th write is $R_i$, then the total number of distinct write sequences is $\prod_{i=1}^{m} 2^{nR_i}$. We conclude that

$$\prod_{i=1}^{m} 2^{nR_i} \leqslant 2^{mnC_{2D}(\alpha,\beta,p)}$$

and, therefore,

$$\frac{\sum_{i=1}^{m} R_i}{m} \leqslant C_{2D}(\alpha, \beta, p).$$

If $m$ goes to infinity, the rate of any $(\alpha, \beta, p)$-constrained code $R$ satisfies

$$R \leqslant C_{2D}(\alpha, \beta, p),$$

i.e., $C(\alpha, \beta, p) \leqslant C_{2D}(\alpha, \beta, p)$. ■

Theorem 2.3.2 provides a scheme to calculate an upper bound on the $(\alpha, \beta, p)$-capacity from an upper bound on the capacity of a two-dimensional constraint. Unfortunately, good upper bounds are known only for some special cases of the values of $\alpha, \beta, p$, and in particular, when $p = 1$. More generally, finding the capacity of a two-dimensional constrained system, such as those mentioned above, is a difficult open problem that has attracted considerable attention over the past 20 years. However, accurate lower and upper bounds on the capacity have been determined for some constraints, as discussed in [65,84,90]. For instance, upper bounds for some square checkerboard constraints are given in [90], from which we can conclude that $C(2, 2, 1) \leqslant 0.43431$ and $C(3, 3, 1) \leqslant 0.25681$.

In the rest of this section we discuss the cases where $\alpha = 1$ or $\beta = 1$. In these cases, the two-dimensional $(a, b, p) = (\alpha, \beta, p)$-constraint of Definition 2.3.1 reduces to a one-dimensional constraint on each row or column, respectively. We consider first the case where $a = 1$, where the corresponding 1-dimensional constraint is described as follows.

**Definition 2.3.3.** *Let $b, p$ be two positive integers. A binary vector $\boldsymbol{u}$ satisfies the* ***$(b, p)$-window-weight-limited (WWL) constraint*** *if for any $b$ consecutive positions there are at most $p$ 1's. We denote the capacity of the constraint by $C_{WWL}(b, p)$.*

According to Theorem 2.3.2, $C_{WWL}(\beta, p)$ is an upper bound on $C(1, \beta, p)$, the capacity of the $(1, \beta, p)$ time-space constraint. Therefore, we are interested in determining the capacity of the general $(b, p)$-WWL constraint. Before addressing this question, we consider some special cases.

We recall the definition of the $(d, k)$-runlength-limited (RLL) constraint, which requires that the number of 0's between adjacent 1's is at least $d$ and at most $k$, and we denote the corresponding capacity by $C_{RLL}(d, k)$. (See, for example, [96], [41].) It is easy to see that the $(b, 1)$-WWL constraint is simply the $(b-1, \infty)$-RLL constraint. Therefore, $C_{RLL}(\beta - 1, \infty)$ is an upper bound on $C(1, \beta, 1)$, a result that was already shown by Jiang et al. [44].

One can also see that the $(b, b-1)$-WWL constraint is the maximum-transition-run MTR$(b-1)$ constraint, which limits the maximum length of a run of 1's to no more than $b-1$ [64]. Interchanging the symbols 0 and 1 establishes a one-to-one correspondence between the MTR$(b-1)$ constraint and the $(0, b-1)$-RLL constraint. Thus, by Theorem 2.3.2, $C_{RLL}(0, \beta - 1)$ is an upper bound on $C(1, \beta, \beta - 1)$.

The capacity of $(d, k)$-RLL constraints is well known and can be elegantly described as the logarithm of the largest real root of a polynomial that depends explicitly on the parameters $d$ and $k$. (We again refer the reader to [96], [41].) However, we have not yet found a comparable formulation of the capacity of the general $(b, p)$-WWL constraint. Therefore, to compute these capacities, we use the general approach that is described in [61].

**Definition 2.3.4.** *A* merge *of two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ of the same length $n$ is a function:*

$$f_n : \{0,1\}^n \times \{0,1\}^n \mapsto \{0,1\}^{n+1} \cup \{\boldsymbol{F}\}.$$

*If the last $n-1$ bits of $\boldsymbol{u}$ are the same as the first $n-1$ bits of $\boldsymbol{v}$, the vector $f_n(\boldsymbol{u}, \boldsymbol{v})$ is the vector $\boldsymbol{u}$ concatenated with the last bit of $\boldsymbol{v}$, otherwise $f_n(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{F}$.*

**Definition 2.3.5.** *Let $b$, $p$ be two positive integers. Let $S_{b,p}$ denote the set of all vectors of length $b-1$ having at most $p$ 1's. That is, $S_{b,p} = \{\boldsymbol{s} \in \{0,1\}^{b-1} : wt(\boldsymbol{s}) \leqslant p\}$. The size of the set $S_{b,p}$ is $M = \sum_{i=0}^{p} \binom{b-1}{i}$. Let $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_M$ be an ordering of the vectors in $S_{b,p}$. The* transition matrix *for the $(b,p)$-WWL constraint, $A_{b,p} = (a_{i,j}) \in \{0,1\}^{M\times M}$, is defined as follows:*

$$a_{i,j} = \begin{cases} 1 & \text{if } f_{b-1}(\boldsymbol{s}_i, \boldsymbol{s}_j) \neq \boldsymbol{F} \text{ and } wt(f_{b-1}(\boldsymbol{s}_i, \boldsymbol{s}_j)) \leqslant p, \\ 0 & \text{otherwise.} \end{cases}$$

**Example 2.3.1.** The following illustrates the construction of the transition matrix $A_{3,2}$ associated with the $(3,2)$-WWL constraint. Note that

$$S_{3,2} = \{\boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{s}_3, \boldsymbol{s}_4\} = \{(0,0), (0,1), (1,0), (1,1)\},$$

The merge of $\boldsymbol{s}_i$ and $\boldsymbol{s}_j$ for $i, j = 1, 2, 3, 4$ determines the matrix $A_{3,2}$. For example, $f_2(\boldsymbol{s}_1, \boldsymbol{s}_1) = (0,0,0)$, $a_{1,1} = 1$; $f_2(\boldsymbol{s}_2, \boldsymbol{s}_1) = \boldsymbol{F}$, $a_{2,1} = 0$; $f_2(\boldsymbol{s}_1, \boldsymbol{s}_2) = (0,0,1)$, $a_{1,2} = 1 \neq a_{2,1}$. This shows that the matrix is not necessarily symmetric. Finally, $f_2(\boldsymbol{s}_3, \boldsymbol{s}_3) = (1,1,1)$, and $a_{3,3} = 0$ since $(1,1,1)$ does not satisfy the (3,2)-WWL constraint.

$$A_{3,2} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}.$$

**Definition 2.3.6.** *A matrix $A \in \{0,1\}^{M\times M}$ is* irreducible *if for all $1 \leqslant i, j \leqslant M$ there exists some $n \geqslant 0$ such that $(A^n)_{i,j} > 0$. Note that $n$ can be a function of $i$ and $j$.*

**Lemma 2.3.7.** *For positive integers $b$, $p$, the transition matrix $A_{b,p}$ is irreducible.*

**Proof.** From the construction of $A_{b,p}$, it is clear that $(A_{b,p}^n)_{i,j}$ is the number of vectors of length $n+b-1$ starting with $\boldsymbol{s}_i$, ending in $\boldsymbol{s}_j$ and satisfying the $(b,p)$-WWL constraint, where $\boldsymbol{s}_i$ and $\boldsymbol{s}_j$ are as described in Definition 2.3.5. Therefore, $A_{b,p}$ is irreducible if for every pair $(i,j)$, there exists a vector of length $n \geqslant 1$ that starts with $\boldsymbol{s}_i$ and ends in $\boldsymbol{s}_j$. Such a vector is obtained by inserting a sufficient number of 0's between $\boldsymbol{s}_i$ and $\boldsymbol{s}_j$. This proves the irreducibility of $A_{b,p}$. ■

Referring to Theorem 3.9 in [61], we have the following characterization of $C_{WWL}(b,p)$.

**Theorem 2.3.8.** *The capacity of the $(b,p)$-WWL constraint is given by*

$$C_{WWL}(b,p) = \log_2(\lambda_{max}),$$

*where $\lambda_{max}$ is the largest real eigenvalue of $A_{b,p}$.*

**Proof.** See Theorem 3.9 in [61]. ■

Figure 2.1 shows $C_{WWL}(\beta, p)$, the upper bound on $C(1, \beta, p)$, for $\beta \leqslant 25$ and $p = 1, 2, 3, 4$. As noted above, the lowest curve corresponds to the capacity of the $(\beta - 1, \infty)$-RLL constraint.

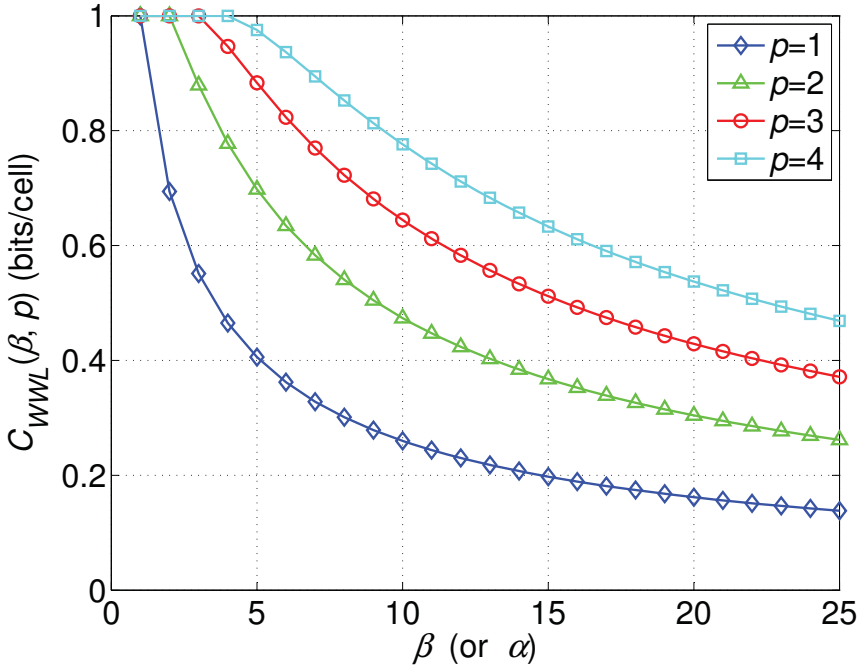


**Figure 2.1**: Upper bound on $C(1, \beta, p)$

**Remark 2.3.1.** The construction of the transition matrix $A_{b,p}$ translates into a graph presentation of the $(b, p)$-WWL constraint in the form of a labeled, directed graph. The states in the graph correspond to the vectors in the set $S_{b,p}$, and the directed edges correspond to the non-zero entries in the matrix $A_{b,p}$. Specifically, if the entry $a_{i,j}$ is non-zero, then there is a directed edge from state $\boldsymbol{s}_i$ to state $\boldsymbol{s}_j$, with label $\boldsymbol{s}_{j,b-1}$, the last bit in $\boldsymbol{s}_j$. Sequences satisfying the $(b, p)$-WWL constraint are generated by reading off the labels along directed paths in the graph. The graph produced by this construction can be identified with a subgraph of the de Bruijn graph on $2^{b-1}$ states. Figure 2.2 illustrates the graph that generates the $(7, 2)$-WWL constraint.

**Remark 2.3.2.** According to Theorem 2.3.2, the capacity of the $(\alpha, p)$-WWL constraint, $C_{WWL}(\alpha, p)$, is an upper bound on the the capacity of the $(\alpha, 1, p)$ time-space constraint,

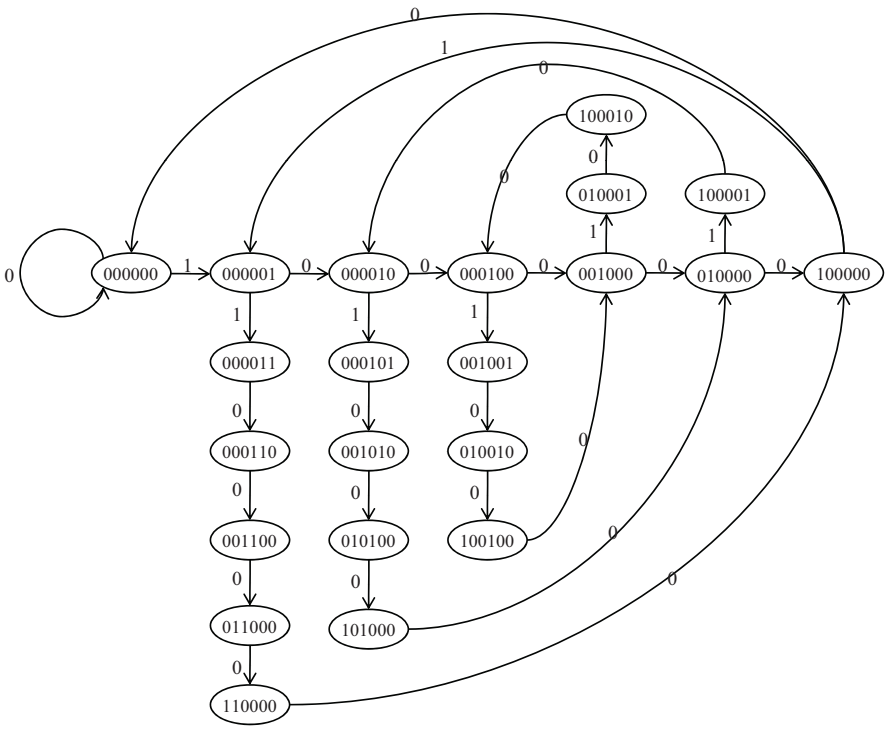


**Figure 2.2**: Labeled graph that generates the $(7, 2)$-WWL constraint

$C(\alpha, 1, p)$. Jiang et al. [44] proposed an upper bound on the rate of an $(\alpha, 1, 1)$-constrained code with fixed block length $n$ and multiple cell levels. In our numerical experiments, their upper bound for binary cells appears to converge to our upper bound as $n \to \infty$.

## 2.4 Lower Bound on the Capacity

In this section, we give lower bounds on the capacity of the $(\alpha, \beta, p)$-constraint based upon specific code constructions. We first present an elementary construction that achieves rate $\frac{p}{\alpha\beta}$. We then show how to improve the bound for the $(1, \beta, p)$- and $(\alpha, 1, p)$-constraints. In this section we assume that for all positive integers $x$ and $y$, the value of $x \pmod{y}$ belongs to the group $\{1, \ldots, y\}$ via the correspondence $\{0, 1, \ldots, y-1\} \to \{y, 1, \ldots, y-1\}$.

The idea of Construction 2.4.1 is to partition the set of $n$ cells into subblocks of size $\beta$. Suppose $p = \beta(q-1) + r$, where $1 \leqslant q \leqslant \alpha$ and $1 \leqslant r \leqslant \beta$. The encoding process has a period of $\alpha$ writes. On the first $q-1$ writes, all cells in each subblock are programmed with no constraint imposed. On the $q$-th write, the first $r$ cells in each subblock are programmed with no constraint and the rest of the cells are not programmed (staying at level 0). From the $(q+1)$-st

write to the $\alpha$-th write, no cells are programmed. The details of the construction are as follows.

**Construction 2.4.1** Let $\alpha, \beta, p$ be positive integers. We construct an $(\alpha, \beta, p)$-constrained code $\mathcal{C}$ of length $n$ as follows. To simplify the construction, we assume that $\beta | n$. Let $q = \left\lceil \frac{p}{\beta} \right\rceil, r = p \pmod{\beta}$, where $1 \leqslant r \leqslant \beta$. For all $i \geqslant 1$, on the $i$-th write, the encoder uses the following rules:

- If $1 \leqslant i \pmod{\alpha} < q$, $n$ bits are written to the $n$ cells.
- If $i \pmod{\alpha} = q$, $rn/\beta$ bits are written in all cells $c_j$ such that $1 \leqslant j \pmod{\beta} \leqslant r$.
- If $i \pmod{\alpha} > q$, no information is written to the cells.

The decoder is implemented in a very similar way.

**Example 2.4.1.** Figure 2.3 shows a typical writing sequence of an $(\alpha = 3, \beta = 3, p = 2)$-constrained code of length 15 based on Construction 2.4.1. The $i$-th row corresponds to the cell-state vector before the $i$-th write. The cells in the box in the $i$-th row are the only cells that can be programmed on the $i$-th write. It can be seen that the rate of the code is the ratio between the number of boxed cells and the total number of cells, which is $\frac{2}{9}$.

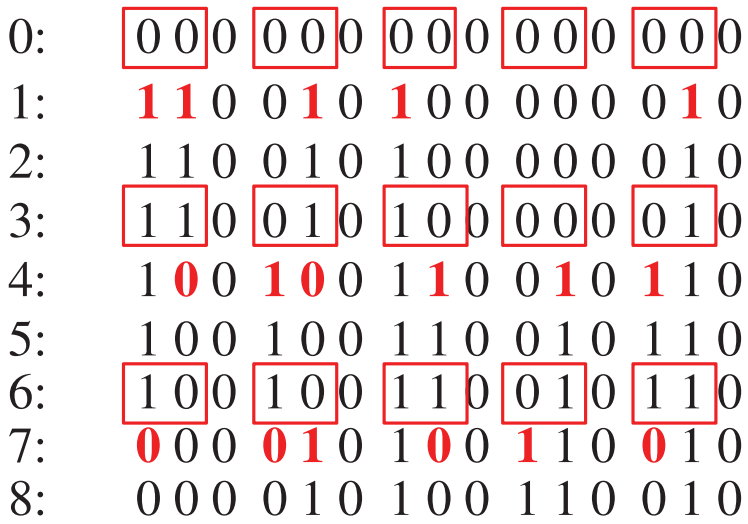


**Figure 2.3**: A sequence of writes of a $(3, 3, 2)$-constrained code

**Theorem 2.4.1.** *The code $\mathcal{C}$ constructed in Construction 2.4.1 is an $(\alpha, \beta, p)$-constrained code and its rate is $R = \frac{p}{\alpha\beta}$.*

**Proof.** We show that for all $i \geqslant 1$ and $1 \leqslant j \leqslant n-\beta+1$, the rewrite cost of the cells $c_j, c_{j+1}, \ldots, c_{j+\beta-1}$ over the writes $i, i+1, \ldots, i+\alpha-1$, is at most $p$. For all $0 \leqslant k \leqslant \alpha-1$ such that $1 \leqslant (i+k) \pmod{\alpha} < q$, all of the $\beta$ cells can be written and since there are $q-1$ such values the rewrite cost on these writes is at most $(q-1)\beta$. For $k$, such that $(i+k) \pmod{\alpha} = q$, at most $r$ out of these $\beta$ cells are programmed and therefore the rewrite cost is at most $r$. For all other values of $k$ no other cells are programmed. Therefore, the total rewrite cost is at most

$$(q-1)\cdot\beta + r = \left(\left\lceil \frac{p}{\beta} \right\rceil - 1\right)\beta + p \pmod{\beta} = p.$$

The total number of bits written on these $\alpha$ writes is $pn/\beta$ and hence the rate of the code is

$$R = \frac{pn/\beta}{\alpha n} = \frac{p}{\alpha\beta}.$$

■

### 2.4.1 Space Constraint Improvement

In this subsection, we improve upon the lower bound on $C(1,\beta,p)$ obtained from the elementary construction. Let $\mathcal{S}_n(b,p)$ be the set of all $(b,p)$-WWL vectors of length $n$. We refer to a subset of $\mathcal{S}_n(b,p)$ as a $(b,p)$**-WWL code** $\mathcal{C}_{WWL}$ of length $n$ . If the size of the code $\mathcal{C}_{WWL}$ is $M$, then it is specified by an encoding map $\mathcal{E}_{WWL} : \{1,\ldots,M\} \mapsto \mathcal{C}_{WWL}$ and a decoding map $\mathcal{D}_{WWL} : \mathcal{C}_{WWL} \mapsto \{1,\ldots,M\}$, such that for all $m \in \{1,\ldots,M\}$, $\mathcal{D}_{WWL}(\mathcal{E}_{WWL}(m)) = m$.

The problem of finding $(b,p)$-WWL codes that approach or achieve the capacity $C_{WWL}(b,p)$ is of independent interest and we address it next. Cover [18] provided an enumerative scheme that can be used to calculate the lexicographic order of any sequence in the constrained system. For the special case of $p=1$, corresponding to RLL block codes, Datta and McLaughlin [19, 20] proposed enumerative methods for binary $(d,k)$-RLL codes based on permutation codes. For $(b,p)$-WWL codes, we find enumerative encoding and decoding strategies with linear complexity enumerating all $(b,p)$-WWL vectors. We present the coding schemes and the complexity analysis in Appendix 6.8. In the sequel, we will simply assume that there exist such codes with rate arbitrarily close to the capacity as the block length goes to infinity for all positive integers $b$ and $p$. The next construction uses $(\beta,p)$-WWL codes to construct $(1,\beta,p)$-constrained codes.

**Construction 2.4.2** Let $\beta, p$ be positive integers such that $p \leqslant \beta$. Let $\mathcal{C}_{WWL}$ be a $(\beta, p)$-WWL code of length $n'$ and size $M$. Let $\mathcal{E}_{WWL}$ and $\mathcal{D}_{WWL}$ be its encoding and decoding maps. A $(1, \beta, p)$-constrained code $\mathcal{C}_{1,\beta,p}$ of length $n = 2n' + \beta - 1$ and its encoding map $\mathcal{E}$ and decoding map $\mathcal{D}$ are constructed as follows.

1. The encoding map $\mathcal{E} : \{1, \ldots, M\} \times \{0,1\}^n \to \{0,1\}^n$ is defined for all $(m, \boldsymbol{u}) \in \{1, \ldots, M\} \times \{0,1\}^n$ to be $\mathcal{E}((m, \boldsymbol{u})) = \boldsymbol{v}$, where
   (a) $\boldsymbol{v}_1^{n'} = \boldsymbol{u}_1^{n'} + \mathcal{E}_{WWL}(m)$,
   (b) $\boldsymbol{v}_{n'+1}^{n'+\beta-1} = \boldsymbol{0}$,
   (c) $\boldsymbol{v}_{n'+\beta}^{n} = \boldsymbol{u}_1^{n'}$,
2. The decoding map $\mathcal{D} : \{0,1\}^n \to \{1, \ldots, M\}$ is defined for all $\boldsymbol{u} \in \{0,1\}^n$ to be

$$\mathcal{D}(\boldsymbol{u}) = \mathcal{D}_{WWL}(\boldsymbol{v}_1^{n'} + \boldsymbol{v}_{n'+\beta}^{n}).$$

**Example 2.4.2.** Here is an example of an $(\alpha = 1, \beta = 3, p = 2)$ code with $n' = 4$ for the first 4 writes. The message set has size $M_{n'} = 13$ (See the definition of $M_{n'}$ in Definition 2.6.1). The length of the memory is $2n' + \beta - 1 = 10$. Suppose on the second write, the message is $m = 7$. Since lexicographically the seventh element in $\mathcal{S}_4(3,2)$ is $(0110)$, the encoder will copy the previous left block $(1011)$ to the right block and flip the second and the third bits in the left block $(1011) \to (1101)$.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1st write, | $m = 11$ : | **1** | 0 | **1** | **1** | 0 | 0 | 0 | 0 | 0 | 0 |
| 2nd write, | $m = 7$ : | 1 | **1** | **0** | 1 | 0 | 0 | **1** | 0 | **1** | **1** |
| 3rd write, | $m = 13$ : | **0** | **0** | 0 | **0** | 0 | 0 | 1 | **1** | **0** | 1 |
| 4th write, | $m = 4$ : | 0 | 0 | **1** | **1** | 0 | 0 | **0** | **0** | 0 | **0** |

**Theorem 2.4.2.** *The code $\mathcal{C}_{1,\beta,p}$ is a $(1, \beta, p)$-constrained code. If the rate of the code $\mathcal{C}_{WWL}$ is $R_{WWL}$, then the rate of the code $\mathcal{C}_{1,\beta,p}$ is $\frac{n'}{2n'+\beta-1} \cdot R_{WWL}$. Both the encoder and decoder of $\mathcal{C}_{1,\beta,p}$ have complexity $O(n)$.*

**Proof.** Let $\boldsymbol{u}$ be the cell-state vector in Construction 2.4.2.

1. For $\boldsymbol{u}_1^{n'}$, encoder step a) guarantees that the positions of rewritten cells satisfy $(\beta, p)$-WWL constraint. So there are at most $p$ reprogrammed cells in any $\beta$ consecutive cells in $\boldsymbol{u}_1^{n'}$.

2. For $\boldsymbol{u}_{n'+\beta}^{n}$, three consecutive writes should be examined. Let $\boldsymbol{w}, \boldsymbol{v}, \boldsymbol{u}$ be the cell-state vectors before the $i$-th, $(i+1)$-st, $(i+2)$-nd writes, $i \geqslant 1$. Encoder step a) means that $\boldsymbol{v}_1^{n'} = \boldsymbol{w}_1^{n'} + \mathcal{E}_{WWL}(m_i)$, where $m_i \in \{1, \ldots, M\}$ is the message to encode on the $i$-th write. Since encoder step c) guarantees that $\boldsymbol{v}_{n'+\beta}^{n} = \boldsymbol{w}_1^{n'}$ and $\boldsymbol{u}_{n'+\beta}^{n} = \boldsymbol{v}_1^{n'}$, we have $\boldsymbol{u}_{n'+\beta}^{n} = \boldsymbol{v}_{n'+\beta}^{n} + \mathcal{E}_{WWL}(m_i)$. This proves that $\boldsymbol{u}_{n'+\beta}^{n}$ satisfies the $(1, \beta, p)$ constraint.

3. For $\boldsymbol{u}_{n'+1}^{n'+\beta-1}$, the cell levels are always set to be 0, which ensures that no violation of the constraint happens between $\boldsymbol{u}_1^{n'}$ and $\boldsymbol{u}_{n'+\beta}^{n}$.

On each write, one of $M$ messages is encoded as a vector of length $n$. Hence, the rate is $\frac{\log_2 M}{n} = \left(\frac{\log_2 M}{n'} \frac{n'}{2n'+\beta-1}\right) = \frac{n'}{2n'+\beta-1} \cdot R_{WWL}$.

The encoder $\mathcal{E}$ and decoder $\mathcal{D}$ come directly from $\mathcal{E}_{WWL}$ and $\mathcal{D}_{WWL}$, which have complexity $O(n)$ both in time and in space. Therefore, $\mathcal{E}$ and $\mathcal{D}$ both have linear complexity in time and in space. ■

**Corollary 2.4.3.** *Let $\beta, p$ be two positive integers such that $p \leqslant \beta$, then*

$$C(1, \beta, p) \geqslant \max\left\{\frac{C_{WWL}(\beta, p)}{2}, \frac{p}{\beta}\right\}.$$

Corollary 2.4.3 provides a lower bound that is achieved by practical coding schemes. In fact, following similar proofs in [4,15,16], we can prove the following theorem using probabilistic combinatorial tools [3].

**Theorem 2.4.4.** *Let $\beta, p$ be positive integers such that $\beta \geqslant p$. Then*

$$C(1, \beta, p) = C_{WWL}(\beta, p).$$

**Proof.** See Appendix 6.9. ■

### 2.4.2 Time Constraint Improvement

Jiang et al. constructed in [44] an $(\alpha, 1, 1)$-constrained code. Let us explain their construction as it serves as the basis for our construction. Their construction uses Write-Once Memory (WOM) codes [71]. A WOM is a storage device consisting of cells that can be used to store any of $q$ values. In the binary case, each cell can be irreversibly changed from state 0 to state 1. We denote by $[n, t; 2^{nR_1}, \ldots, 2^{nR_t}]$ a $t$-write WOM code $\mathcal{C}_W$ such that the number of messages

that can be written to the memory on its $i$-th write is $2^{nR_i}$, and the sum-rate of the WOM code is defined to be $R_{\text{sum}} = \sum_{i=1}^{t} R_i$. The sum-capacity $C_{\text{sum}}$ is defined as the supremum of achievable sum-rates. The code is specified by $t$ pairs of encoding and decoding maps, $(\mathcal{E}_i, \mathcal{D}_i)$, where $i \in \{1, 2, \ldots, t\}$. Assuming that the cell-state vector before the $i$-th write is $\boldsymbol{c}_i$, the encoder is a map

$$\mathcal{E}_i : [1 : 2^{nR_i}] \times \{0,1\}^n \to \{0,1\}^n,$$

such that for all $(m, \boldsymbol{c}_{i-1}) \in [1 : 2^{nR_i}] \times \{0,1\}^n$,

$$\boldsymbol{c}_{i-1} \preceq \boldsymbol{c}_i = \mathcal{E}_i(m, \boldsymbol{c}_{i-1}),$$

where the relation "$\preceq$" is defined in Definition 2.6.1. The decoder

$$\mathcal{D}_i : \{0,1\}^n \to [1 : 2^{nR_i}],$$

satisfies

$$\mathcal{D}_i(\mathcal{E}_i(m, \boldsymbol{c}_{i-1})) = m.$$

for all $m \in [1 : 2^{nR_i}]$,

It has been shown in [36] that the sum-capacity of a $t$-write WOM is $C_{\text{sum}} = \log_2(t + 1)$.

**Example 2.4.3.** Table 2.1 [71] shows the encoding and decoding maps of a 2-write WOM code using 3 cells, where on each write, 2 bits are written. Suppose all cells are initiated as 0. If the written messages are 1 and 3 on the first and the second write, respectively, then the cell-state vector is changed as $(000) \to (001) \to (011)$; if on both write, message 2 is written, then the cell-state vector is changed as $(000) \to (010) \to (010)$.

**Table 2.1**: A 3-cell 2-write WOM code

| message | 1st write | 2nd write |
|---|---|---|
| 0 | 000 | 111 |
| 1 | 001 | 110 |
| 2 | 010 | 101 |
| 3 | 100 | 011 |

The constructed $(\alpha, 1, 1)$-constrained code has a period of $2(t + \alpha)$ writes. On the first $t$ writes of each period, the encoder simply writes the information using the encoding maps of the $t$-write WOM code. Then, on the $(t + 1)$-st write, no information is written but all the cells are increased to level one. On the following $\alpha - 1$ writes no information is written and the cells

do not change their levels; that completes half of the period. On the next $t$ writes the same WOM code is again used; however since now all the cells are in level one, the complement of the cell-state vector is written to the memory on each write. On the next write no information is written and the cells are reduced to level zero. In the last $\alpha - 1$ writes no information is written and the cells do not change their values. We present this construction now in detail.

**Construction 2.4.3** Let $\alpha$ be a positive integer and let $\mathcal{C}_W$ be an $[n, t; 2^{nR_1}, \ldots, 2^{nR_t}]$ $t$-write WOM code. Let $\mathcal{E}_i(m, \boldsymbol{v}_{i-1})$ be the $i$-th encoder of $\mathcal{C}_W$, for $m \in [1 : 2^{nR_i}], i \in [1 : t]$. An $(\alpha, 1, 1)$-constrained code $\mathcal{C}_{\alpha,1,1}$ is constructed as follows. For all $i \geqslant 1$, let $i' = i \pmod{2(t+\alpha)}$, where $1 \leqslant i' \leqslant 2(t+\alpha)$. The cell-state vector after the $i$-th write is denoted by $\boldsymbol{c}_i$. On the $i$-th write, the encoder uses the following rules:

- If $i' \in [1 : t]$, write $M_{i'} \in [1 : 2^{nR_{i'}}]$ such that
$$\boldsymbol{c}_i = \mathcal{E}_{i'}(M_{i'}, \boldsymbol{c}_{i-1}).$$
- If $i' = t + 1$, no information is written and the cell-state vector is changed to the all-one vector $\mathbf{1}$, i.e., $\boldsymbol{c}_i = \mathbf{1}$.
- If $i' \in [t + 2 : t + \alpha]$, no information is written and the cell-state vector is not changed.
- If $i' \in [t + \alpha + 1 : 2t + \alpha]$, write $M_{i'-t-\alpha} \in [1 : 2^{nR_{i'-t-\alpha}}]$ such that
$$\boldsymbol{c}_i = \overline{\mathcal{E}_{i'-t-\alpha}(M_{i'-t-\alpha}, \overline{\boldsymbol{c}}_{i-1})}.$$
- If $i' = 2t + \alpha + 1$, no information is written and the cell-state vector is changed to the all-zero vector $\mathbf{0}$, i.e., $\boldsymbol{c}_i = \mathbf{0}$.
- If $i' \in [2t + \alpha + 1 : 2(t + \alpha)]$, no information is written and the cell-state vector is not changed.

**Remark 2.4.1.** This construction is presented differently in [44]. This results from the constraint of having the same rate on each write which we can bypass in this work. Consequently, in our case we can have varying rates and thus the code $\mathcal{C}_{\alpha,1,p}$ can achieve a higher rate.

**Theorem 2.4.5.** *The code $\mathcal{C}_{\alpha,1,1}$ is an $(\alpha, 1, 1)$-constrained code. If the $t$-write WOM code $\mathcal{C}_W$ is sum-rate optimal, then the rate of $\mathcal{C}_{\alpha,1,1}$ is $\frac{\log_2(t+1)}{t+\alpha}$.*

**Proof.** In every period of $2(t+\alpha)$ writes, every cell is programmed at most twice; once in the first $t+1$ writes and once in the first $t+1$ writes of the second part of the write-period. After every sequence $t+1$ writes, the cell is not programmed for $\alpha-1$ writes. Therefore the rewrite cost of every cell among $\alpha$ consecutive rewrites is at most 1.

If the rate of the WOM code $\mathcal{C}_W$ is $R_W$ then $2nR_W$ bits are written in every period of $2(t+\alpha)$ writes. Hence, the rate of $\mathcal{C}_{\alpha,1,1}$ is $\frac{2nR_W}{2(t+\alpha)n} = \frac{R_W}{t+\alpha}$. If $\mathcal{C}_W$ is sum-rate optimal, the rate of $\mathcal{C}_{\alpha,1,1}$ is therefore $\frac{\log_2(t+1)}{t+\alpha}$. ■

Table 2.2 shows the highest rates of $(\alpha,1,1)$-constrained codes based on Construction 2.4.3 for $\alpha = 4,\ldots,8$.

**Table 2.2**: Highest rates of $(\alpha,1,1)$-constrained codes

| $\alpha$ | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|
| $1/\alpha$ | 0.25 | 0.2 | 0.167 | 0.143 | 0.125 |
| rate of $\mathcal{C}_{\alpha,1,1}$ | 0.290 | 0.256 | 0.235 | 0.216 | 0.201 |

Next, we would like to extend Construction 2.4.3 in order to construct $(\alpha,1,p)$-constrained codes for all $p \geqslant 2$. For simplicity of the construction, we will assume that $p$ is an even integer; and the required modification for odd values of $p$ will be immediately clear. We choose $t \geqslant 1$ such that $\alpha \geqslant (p-1)t$ and the period of the code is $\alpha+t$. On the first $t$ writes of each period, the encoder uses the encoding map of the $t$-write WOM code. In the following $t$ writes, it uses the bit-wise complement of a WOM code as in Construction 2.4.3. This procedure is repeated for $\frac{p}{2}$ times; this completes the first $tp$ writes in the period. On the $(tp+1)$-st write, no new information is written and the cell-state vector is changed to the all-zero vector. During the $(tp+2)$-nd to $(\alpha+t)$-th writes, no information is written and the cell-state vector is not changed. That completes one period of $\alpha+t$ writes.

**Remark 2.4.2.** If $p$ is odd, then on the $(tp+1)$-st write, no new information is written and the cell-state vector is changed to the all-one vector. It is not changed until the $(\alpha+t)$-th write to complete a period. Now the cell-state vector is an all-one vector. For the next period of $(\alpha+t)$ writes, the encoder uses the bit-wise complement of the first period and the cell-state vector returns to all-zero state afterwards.

**Construction 2.4.4** Let $\alpha, p, t$ be positive integers such that $\alpha \geqslant (p-1)t$ and $p$ is even. Let $\mathcal{C}_W$ be an $[n,t;2^{nR_1},\ldots,2^{nR_t}]$ $t$-write WOM code. For $i \in [1:t]$, let $\mathcal{E}_i(m,\boldsymbol{v}_{i-1})$ be its

encoding map on the $i$-th write, where $m \in [1 : 2^{nR_i}]$. An $(\alpha, 1, p)$-constrained code $\mathcal{C}_{\alpha,1,p}$ is constructed as follows. For all $i \geqslant 1$, let $i' = i \pmod{\alpha + t}$, $i'' = i' \pmod{2t}$ where $1 \leqslant i' \leqslant (\alpha + t), 1 \leqslant i'' \leqslant 2t$. The cell-state vector after the $i$-th write is denoted by $\boldsymbol{c}_i$. On the $i$-th write, the encoder uses the following rules:

- If $i' \in [1 : pt]$ and $i'' \in [1 : t]$, write $M_{i''} \in [1 : 2^{nR_{i''}}]$ such that

$$\boldsymbol{c}_i = \mathcal{E}_{i''}(M_{i''}, \boldsymbol{c}_{i-1}).$$

- If $i' \in [1 : pt]$ and $i'' \in [t + 1 : 2t]$, write $M_{i''-t} \in [1 : 2^{nR_{i''-t}}]$ such that

$$\boldsymbol{c}_i = \overline{\mathcal{E}_{i''-t}(M_{i''-t}, \overline{\boldsymbol{c}}_{i-1})}.$$

- If $i' = pt + 1$, no information is written and the cell-state vector is changed to $\mathbf{0}$, i.e., $\boldsymbol{c}_i = \mathbf{0}$.

- If $i' \in [pt + 2 : \alpha + t]$, no information is written and the cell-state vector is not changed.

**Example 2.4.4.** Suppose $\alpha = 3, p = 2$ and $t = 2$ in Construction 2.4.4 and $\mathcal{C}_W$ is the WOM code in Example 2.4.3. The period of Construction 2.4.4 is $\alpha + t = 5$. Suppose all cells are initiated as 0 and the messages to write is $(1, 3, 2, 1)$ on the first 4 writes, and no information is written on the fifth write, then the cell-state vector is changed as $(000) \xrightarrow{1} (001) \xrightarrow{2} (011) \xrightarrow{3} (101) \xrightarrow{4} (001) \xrightarrow{5} (000)$.

**Theorem 2.4.6.** *The code $\mathcal{C}_{\alpha,1,p}$ is an $(\alpha, 1, p)$-constrained code. If the $t$-write WOM code $\mathcal{C}_W$ is sum-rate optimal, then the rate of $\mathcal{C}_{\alpha,1,p}$ is $\frac{p \log_2(t+1)}{\alpha+t}$.*

**Proof.** This is similar to the proof of Theorem 2.4.5, so we present here only a sketch of the proof. In every period of $(\alpha + t)$ writes, each cell is rewritten at most $p$ times. In particular, the first rewrite happens before the $(t + 1)$-st write. After that, the cell is rewritten at most $p - 1$ times until the $(tp + 1)$-st write and then not programmed for $\alpha + t - (tp + 1)$ writes. Therefore, each cell is rewritten at most $p$ times on $\alpha + t - (tp + 1) + (tp + 1) - t = \alpha$ writes. This proves the validity of the code.

If the rate of the WOM code $\mathcal{C}_W$ is $R_W$ then $pnR_W$ bits are written during each period of $\alpha + t$ writes since the WOM code is used $p$ times. Hence, the rate of $\mathcal{C}_{\alpha,1,p}$ is $\frac{2pnR_W}{2(\alpha+t)n} = \frac{pR_W}{\alpha+t}$. If that $\mathcal{C}_W$ is sum-rate optimal, the rate of $\mathcal{C}_{\alpha,1,p}$ is $\frac{p \log_2(t+1)}{\alpha+t}$. ■

**Remark 2.4.3.** In Construction 2.4.4 we required that $\alpha \geqslant (p-1)t$ and, in particular, $t \leqslant \left\lfloor \frac{\alpha}{p-1} \right\rfloor$. If $t > \left\lfloor \frac{\alpha}{p-1} \right\rfloor$, we can simply use Construction 2.4.4 while taking $\alpha = (p-1)t$, i.e., the period of writes is now $pt$ and and we construct a $((p-1)t, 1, p)$-constrained code, which is also an $(\alpha, 1, p)$-constrained code. The rate of the code is $R_W/t$, where $R_W$ is the rate of the WOM code $\mathcal{C}_W$.

The next corollary provides lower bounds on $C(\alpha, 1, p)$.

**Corollary 2.4.7.** *Let $\alpha, p$ be positive integer such that $p \leqslant \alpha$. Then,*

$$C(\alpha, 1, p) \geqslant \max_{t, t^* \in \mathbb{Z}_+,} \left\{ \frac{p \log_2(t+1)}{\alpha + t}, \frac{\log_2(t^*+1)}{t^*}, \frac{p}{\alpha} \right\},$$

*where*

$$1 \leqslant t \leqslant \left\lfloor \frac{\alpha}{p-1} \right\rfloor, t^* = \left\lceil \frac{\alpha}{p-1} \right\rceil.$$

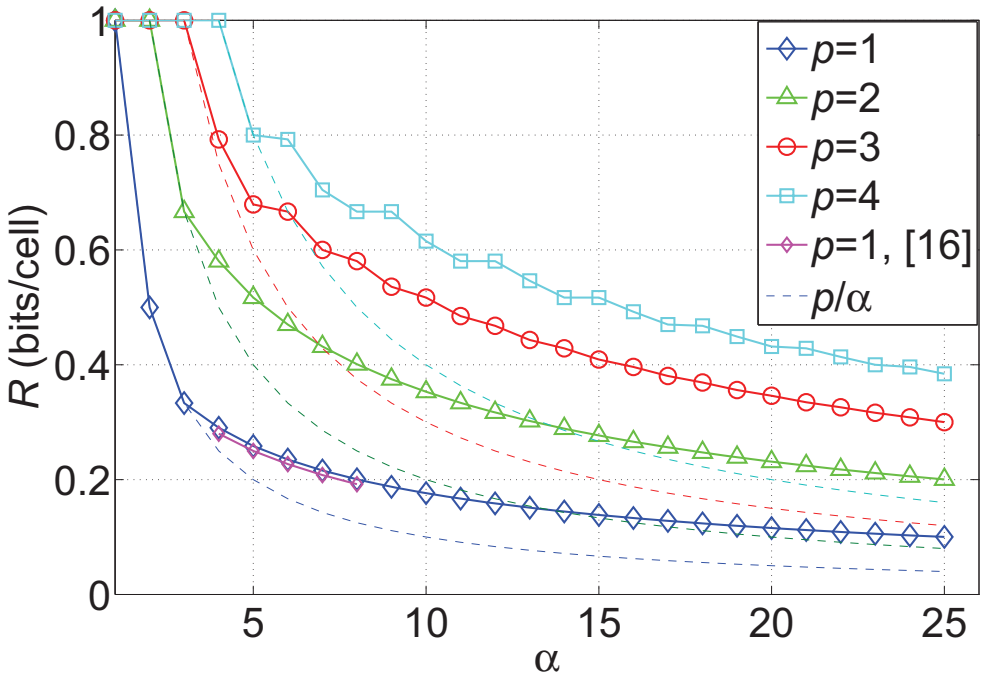


**Figure 2.4**: Lower bound on $C(\alpha, 1, p)$

Figure 2.4 shows the rates of $(\alpha, \beta = 1, p)$ constrained codes obtained by selecting the best $t$ for each pair of $(\alpha, p)$. Note that the curve for $p = 1$ is obtained by implementing the ideas in [44] and there is a slight improvement over the 5 points which is the rates listed in [44] for $\alpha = 4, \ldots, 8$, the reason to which is discussed in Remark 2.4.1. In comparison to the codes

in Construction 2.4.1 whose rates are shown by the dashed lines, our construction approximately doubles the rates. Our lower bounds achieve approximately 78% of the corresponding upper bounds on $C(\alpha, 1, p)$.

### 2.4.3 Time-Space Constraint Improvement

In this section, we are interested in combining the improvements in time and in space to provide lower bounds on the capacity of $(\alpha, \beta, p)$-constraints.

**Theorem 2.4.8.** *For all $\alpha, \beta, p$ positive integers,*

$$C(\alpha, \beta, p) \geqslant \max\left\{\frac{C(\alpha, 1, p)}{\beta}, \frac{C(1, \beta, p)}{\alpha}\right\}.$$

**Proof.** An $(\alpha, \beta, p)$-constrained code can be constructed in two ways.

1. Let $\mathcal{C}$ be a $(1, \beta, p)$-constrained code of rate $R$ and length $n$. We construct a new code $\mathcal{C}'$ with the same number of cells. New information is written to the memory on all $i$-th writes, where $i \equiv 1 \pmod{\alpha}$, simply by using the $\left\lceil \frac{i}{\alpha} \right\rceil$-th write of the code $\mathcal{C}$. Then, the code $\mathcal{C}'$ is an $(\alpha, \beta, p)$-constrained code and its rate is $R/\alpha$. Therefore, we conclude that $C(\alpha, \beta, p) \geqslant \frac{C(1,\beta,p)}{\alpha}$.

2. Let $\mathcal{C}$ be an $(\alpha, 1, p)$-constrained code of rate $R$ and length $n$. We construct a new code $\mathcal{C}'$ for $n\beta$ cells: $(c_1, c_2, \ldots, c_{n\beta})$. The code $\mathcal{C}'$ uses the same encoding and decoding maps of the code $\mathcal{C}$, while using only the $n$ cells $c_i$ such that $i \equiv 1 \pmod{\beta}$. Then, the code $\mathcal{C}'$ is an $(\alpha, \beta, p)$-constrained code and its rate is $R/\beta$. Therefore, we conclude that $C(\alpha, \beta, p) \geqslant \frac{C(\alpha,1,p)}{\beta}$.

The capacity must be greater than or equal to the maximum of the two lower bounds. ■

## 2.5 Conclusion

In this chapter, we study the time-space constraint for PCM, which was originally proposed in [44]. A code is called an $(\alpha, \beta, p)$-constrained code if for any $\alpha$ consecutive rewrites and for any segment of $\beta$ contiguous cells, the total rewrite cost of the $\beta$ cells over those $\alpha$ rewrites is at most $p$. Here, the cells are binary and the rewrite cost is defined to be the Hamming distance between the current and next memory states. First, we show a general upper bound on the achievable rate of these codes which extends the results of Jiang et al. Then, we

generalize their construction for $(\alpha \geqslant 1, \beta = 1, p = 1)$-constrained codes and show another construction for $(\alpha = 1, \beta \geqslant 1, p \geqslant 1)$-constrained codes. Finally, we show that these two constructions can be used to construct codes for all values of $\alpha$, $\beta$, and $p$.

## 2.6 Appendix A

In this section, we show an enumerative encoding and decoding strategy with linear complexity for the set of $(\beta, p)$-WWL vectors.

**Definition 2.6.1.** *Let $\mathbf{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ be a set of distinct binary vectors, $\boldsymbol{x}_i \in \{0,1\}^n, i = 1, \ldots, N$. Let $\psi(\boldsymbol{x})$ denote the decimal representation of a vector $\boldsymbol{x} \in \{0,1\}^n$. For $\boldsymbol{x}, \boldsymbol{y} \in \{0,1\}^n$, we say $\boldsymbol{x} \preceq \boldsymbol{y}$ (or $\boldsymbol{x} \prec \boldsymbol{y}$) if and only if $\psi(\boldsymbol{x}) \leqslant \psi(\boldsymbol{y})$ (or $\psi(\boldsymbol{x}) < \psi(\boldsymbol{y})$). The* order *of the element $\boldsymbol{x}_i$ in $\mathbf{X}$ is defined as:*

$$ord(\boldsymbol{x}_i) = \big|\{j : \boldsymbol{x}_j \preceq \boldsymbol{x}_i, 1 \leqslant j \leqslant N\}\big|.$$

Let $\{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_{M_n}\}$ be an ordering of the elements in $\mathcal{S}_n(\beta, p)$, where $M_n = |\mathcal{S}_n(\beta, p)|$. The encoder and decoder of a $(\beta, p)$-WWL code give a one-to-one mapping between $\mathcal{S}_n(\beta, p)$ and $\{1, \ldots, M_n\}$, namely $\mathcal{E}_{WWL}(m) = \boldsymbol{c}_m$ where $ord(\boldsymbol{c}_m) = m$ and $\mathcal{D}_{WWL}(\boldsymbol{c}_m) = ord(\boldsymbol{c}_m) = m$, for all $m = \{1, \ldots, M_n\}$. Now the problem is to calculate $ord(\boldsymbol{c}_m)$ given $\boldsymbol{c}_m$. Let $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{M_{\beta-1}}$ be the ordering of the vectors in $S_{\beta,p}$ introduced in Definition 2.3.5, where $M_{\beta-1} = |S_{\beta,p}| = |\mathcal{S}_{\beta-1}(\beta, p)| = \sum_{i=0}^{p} \binom{\beta-1}{i}$. Let

$$\boldsymbol{x}_{\beta,p,n} = (x_1(n), x_2(n), \ldots, x_{M_{\beta-1}}(n))^T,$$

where $x_i(n)$ is the number of $(\beta, p)$-WWL vectors of length $n$ that have the vector $\boldsymbol{s}_i$ as a prefix, where $\boldsymbol{x}^T$ denotes the transpose of $\boldsymbol{x}$.

**Lemma 2.6.2.** *The vectors $\boldsymbol{x}_{\beta,p,n+1}$, $n \geqslant \beta$, satisfy the first-order recursion:*

$$\boldsymbol{x}_{\beta,p,n+1} = A_{\beta,p} \cdot \boldsymbol{x}_{\beta,p,n}.$$

**Proof.** See [90]. ■

The encoder and decoder have access to a matrix $\boldsymbol{X}_{\beta,p,n} \in \mathbf{Z}_{+}^{(n+\beta)\times M_{\beta-1}}$, where the $i$-th row of $\boldsymbol{X}_{\beta,p,n}$ is $\boldsymbol{x}_{\beta,p,i}^T$, $i = 1, \ldots, n+\beta$. For simplicity, $\boldsymbol{X}_{\beta,p,n}$ is written as $\boldsymbol{X}$ if no confusion can occur. We denote by $\boldsymbol{X}(i,j)$ the entry in the $i$-th row and $j$-th column of $\boldsymbol{X}$ and we define $\boldsymbol{X}(i,:)$, $\boldsymbol{X}(:,j)$ to be the $i$-th row vector, $j$-th column vector of $\boldsymbol{X}$, respectively,

i.e., $\boldsymbol{X}(i,:) = (\boldsymbol{X}(i,1),\ldots,\boldsymbol{X}(i,M_{\beta-1}))$ and $\boldsymbol{X}(:,j) = (\boldsymbol{X}(1,j),\ldots,\boldsymbol{X}(n+\beta,j))^T$. From Lemma 2.6.2, $\boldsymbol{X}_{\beta,p,n}$ can be calculated efficiently with time complexity $O(n)$.

**Decoder**

Based on $\boldsymbol{X}_{\beta,p,n}$, we present an enumerative method to calculate the order of each element in $\mathcal{S}_n(\beta,p)$. Note that the order of a vector is the decoded message corresponding to that vector. In this algorithm, the decoder scans the vector from left to right. Whenever the decoder finds a 1 in the vector, the order of the vector will increase. The details of the algorithm are presented below. Here $\boldsymbol{c} = (c_1,\ldots,c_n) \in \mathcal{S}_n(\beta,p)$ is the binary vector to be decoded; the algorithm calculates $ord(\boldsymbol{c}) \in \{1,\ldots,M_n\}$.

**Algorithm 2.6.3** DECODING: CALCULATE $ord(\boldsymbol{c})$, $\boldsymbol{c} \in \mathcal{S}_n(\beta,p)$

1: let $cnt = 0, j = 1, i = 0$;
2: while $(i \leqslant n)\{$
3: while $(j \leqslant n$ and $c(j) \neq 1)$
4: $j = j + 1$;
5: if $(j = n+1)$
6: $ord(\boldsymbol{c}) = cnt + 1$;
7: algorithm ends;
8: $\}$
9: /*A 1 is detected in $\boldsymbol{c}$.*/
10: let $\boldsymbol{d} = (0,\ldots,0)$ with length $\beta - 1$;
11: /*$\boldsymbol{d}$ is a vector storing $\beta - 2$ bits to the left of the detected 1, with a 0 appended.*/
12: if $(j \geqslant \beta - 1)$
13: let $\boldsymbol{d}_1^{\beta-2} = \boldsymbol{c}_{j-\beta+2}^{j-1}$;
14: else /*$j < \beta - 1$*/
15: let $\boldsymbol{d}_{\beta-j}^{\beta-2} = \boldsymbol{c}_1^{j-1}$;
16: find $k \in [1 : M_{\beta-1}]$ such that $\boldsymbol{s}_k = \boldsymbol{d}$;
17: $cnt = cnt + \boldsymbol{X}(n-j+\beta-1,k)$;
18: $i = j$; $j = i+1$;
19: $\}$

20: $ord(\boldsymbol{c}) = cnt + 1$;

21: *algorithm ends.*

**Example 2.6.1.** Suppose we would like to decode a $(6,3)$-WWL vector $\boldsymbol{c} = (1011001001)$ of length 10.

- A 1 is detected $(\mathbf{1}011001001)$, where $i = 0, j = 1$. The decoder aims to find the number of vectors $\hat{\boldsymbol{c}}$ such that $(0000000000) \preceq \hat{\boldsymbol{c}} \prec (1000000000)$. Now $\boldsymbol{d} = (00000) = \boldsymbol{s}_1$, so $k = 1$, and $n - j + \beta - 1 = 14$. Therefore, $cnt = 0 + \boldsymbol{X}_{6,3,16}(14,1) = 236$.

- A 1 is detected $(10\mathbf{1}1001001)$, where $i = 1, j = 3$. The decoder aims to find the number of vectors $\hat{\boldsymbol{c}}$ such that $(1000000000) \preceq \hat{\boldsymbol{c}} \prec (1010000000)$. Here $\boldsymbol{d} = (00100) = s_5$, so $k = 5$, and $n - j + \beta - 1 = 12$. Therefore, $cnt = 236 + \boldsymbol{X}_{6,3,16}(12,5) = 308$.

- A 1 is detected $(101\mathbf{1}001001)$, where $i = 3, j = 4$. The decoder aims to find the number of vectors $\hat{\boldsymbol{c}}$ such that $(1001000000) \preceq \hat{\boldsymbol{c}} \prec (1011000000)$. Here $\boldsymbol{d} = (01010) = s_{11}$, so $k = 11$, and $n - j + \beta - 1 = 11$. Therefore, $cnt = 308 + \boldsymbol{X}_{6,3,16}(11,11) = 343$.

- A 1 is detected $(1011001\mathbf{1}001)$, where $i = 4, j = 7$. The decoder aims to find the number of vectors $\hat{\boldsymbol{c}}$ such that $(1011000000) \preceq \hat{\boldsymbol{c}} \prec (1011001000)$. Here $\boldsymbol{d} = (11000) = s_{23}$, so $k = 23$, and $n - j + \beta - 1 = 8$. Therefore, $cnt = 343 + \boldsymbol{X}_{6,3,16}(8,23) = 351$.

- Finally, a 1 is detected $(101100100\mathbf{1})$, where $i = 7, j = 10$. The decoder aims to find the number of vectors $\hat{\boldsymbol{c}}$ such that $(1011001000) \preceq \hat{\boldsymbol{c}} \prec (1011001001)$. Here $\boldsymbol{d} = (01000) = s_9$, so $k = 9$, and $n - j + \beta - 1 = 5$. Therefore, $cnt = 351 + \boldsymbol{X}_{6,3,16}(5,9) = 352$.

We calculate that $ord(\boldsymbol{c}) = cnt + 1 = 353$ and $\boldsymbol{c}$ is decoded as 353.

**Theorem 2.6.4.** *Algorithm* 2.6.3 *calculates the order of a* $(\beta, p)$*-WWL vector of length* $n$ *in* $\mathcal{S}_n(\beta, p)$*. Its time complexity and space complexity are both* $O(n)$*.*

**Proof.** We first show the correctness of the algorithm and then analyze its time and space complexity.

Correctness: Let $\boldsymbol{c}$ be the vector to decode; that is, we seek to find $ord(\boldsymbol{c})$. For $\boldsymbol{c}_1 \preceq \boldsymbol{c}_2$, we denote by $N(\boldsymbol{c}_1, \boldsymbol{c}_2)$ the number of vectors $\hat{\boldsymbol{c}}$ such that $\boldsymbol{c}_1 \preceq \hat{\boldsymbol{c}} \prec \boldsymbol{c}_2$. Let $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_L$ be a sequence of vectors such that $\mathbf{0} = \boldsymbol{c}_0 \preceq \boldsymbol{c}_1 \preceq \boldsymbol{c}_2 \preceq \cdots \preceq \boldsymbol{c}_L = \boldsymbol{c}$; then it is easy to see

$$ord(\boldsymbol{c}) = \sum_{i=1}^{L} N(\boldsymbol{c}_{i-1}, \boldsymbol{c}_i) + 1.$$

Let $L$ be the number of 1's in $\boldsymbol{c}$; let all the indices of 1's be $j_1, j_2, \ldots, j_L$ in ascending order, that is $1 \leqslant j_1 < \cdots < j_L \leqslant n$ and $c_{j_1} = c_{j_2} = \cdots = c_{j_L} = 1$. For $i \in \{1, \ldots, L\}$, $\boldsymbol{c}_i$ is chosen such that $\boldsymbol{c}_i = \boldsymbol{c}_{i-1} + \boldsymbol{\delta}_{j_i}$, where $\boldsymbol{c}_0 = \boldsymbol{0}$, and $\boldsymbol{\delta}_j$, $j \in \{1, \ldots, n\}$, denotes the vector where all entries are 0 except for the $j$-th entry, which is a 1. Here addition is component-wise modulo-2 summation.

Lines 3 and 4 together with Line 18 in Algorithm 2.6.3 scan $\boldsymbol{c}$ and find $\boldsymbol{c}_i$ according to $\boldsymbol{c}_{i-1}$. Therefore, we are left to prove that Algorithm 2.6.3 calculates $N(\boldsymbol{c}_{i-1}, \boldsymbol{c}_i)$ for $i \in \{1, \ldots, L\}$.

By definition, the first $j_i - 1$ digits of $\boldsymbol{c}_i$ and $\boldsymbol{c}_{i-1}$ are the same, and $c_{i,j_i} = 1$ while $c_{i-1,j_i} = 0$. Then a vector $\hat{\boldsymbol{c}} \in \{0,1\}^n$ satisfies $\boldsymbol{c}_{i-1} \preceq \hat{\boldsymbol{c}} \prec \boldsymbol{c}_i$ if and only if the first $j_i$ digits of $\hat{\boldsymbol{c}}$ are the same as those of $\boldsymbol{c}_{i-1}$, i.e. $\hat{\boldsymbol{c}}_1^{j_i} = \boldsymbol{c}_{i-1,1}^{j_i}$. Given the length and the first $j_i$ digits of $\hat{\boldsymbol{c}}$, the number of possible $\hat{\boldsymbol{c}}$ can be calculated based on the matrix $\boldsymbol{X}$ in the following way. Since the $(\beta, p)$-WWL constraint is local, if $j_i > \beta - 1$, the task is equivalent to calculating the number of $\tilde{\boldsymbol{c}}$ with length $n - j_i + \beta - 1$ such that the first $\beta - 1$ digits are a prefix of $\hat{\boldsymbol{c}}$, in particular, $\tilde{\boldsymbol{c}}_1^{\beta-1} = \hat{\boldsymbol{c}}_{j_i-\beta+2}^{j_i}$; otherwise, for $j_i \leqslant \beta - 1$, it is equivalent to calculating the number of $\tilde{\boldsymbol{c}}$ with length $n - j_i + \beta - 1$ such that the first $\beta - 1$ digits are zeros followed by length-$j_i$ prefix of $\hat{\boldsymbol{c}}$, that is, $\tilde{\boldsymbol{c}}_1^{\beta-1} = (\boldsymbol{0}_{\beta-1-j_i}, \hat{\boldsymbol{c}}_1^{j_i})$. Lines 10 – 15 in Algorithm 2.6.3 find the first $\beta - 1$ digits of $\tilde{\boldsymbol{c}}$ and Lines 16 and 17 calculate the number of $\tilde{\boldsymbol{c}}$, which is the number of vectors $\hat{\boldsymbol{c}}$ satisfying $\boldsymbol{c}_{i-1} \preceq \hat{\boldsymbol{c}} \prec \boldsymbol{c}_i$. Therefore, Algorithm 2.6.3 calculates $N(\boldsymbol{c}_{i-1}, \boldsymbol{c}_i)$ for $i \in \{1, \ldots, L\}$ and sums them up to derive the order of $\boldsymbol{c}$.

Time complexity analysis: It can be seen from the algorithm that the decoder scans the vector that is to be decoded only once. Whenever the decoder detects a 1, it uses binary searches to find the corresponding prefix vector $\boldsymbol{d}$ in $\boldsymbol{X}$, while the number of 1's is no more than $\frac{np}{\beta}$. Therefore, the time complexity of the decoder is no more that $O(\frac{np}{\beta} \log M_{\beta-1}) = O(\frac{np}{\beta} \log \sum_{i=0}^{p} \binom{\beta-1}{i}) = O(n)$, where $\beta$ and $p$ are fixed integers and not related to $n$.

Space complexity analysis: The space complexity comes from the matrix $\boldsymbol{X}$ with $n + \beta - 1$ rows and $M_{\beta-1}$ columns. Therefore, the space complexity is also $O(n)$ since $\beta$ and $M_{\beta-1}$ are both fixed integers. ■

**Encoder**

The encoder follows a similar approach to map an integer $m \in \{1, \ldots, M_n\}$ to a vector $\boldsymbol{c} \in \mathcal{S}_n(\beta, p)$, such that $ord(\boldsymbol{c}) = m$. We call $\boldsymbol{c}$ the encoded vector for the message $m$. Note that

$\forall m_i, m_j \in \{1, \ldots, M_n\}, m_i \leqslant m_j$ if and only if $\boldsymbol{c}_i \preceq \boldsymbol{c}_j$, where $ord(\boldsymbol{c}_i) = m_i$ and $ord(\boldsymbol{c}_j) = m_j$. The following encoding algorithm uses the matrix $\boldsymbol{X}$ to efficiently calculate the vector $\boldsymbol{c} \in \mathcal{S}_n(\beta, p)$ such that $ord(\boldsymbol{c}) = m$, for $m \in \{1, \ldots, M_n\}$. The algorithm has linear complexity.

**Algorithm 2.6.5** ENCODING: FIND $\boldsymbol{c}$ SUCH THAT $ord(\boldsymbol{c}) = m$

*let* $cnt = 0, \boldsymbol{c} = (0, \ldots, 0)$ *with length* $n$*;*

*for* $i = 1, 2, \ldots, n$ {

*let* $\boldsymbol{t} = \boldsymbol{c}$*;*

*let* $t(i) = 1$*;*

*if* $\boldsymbol{t}$ *satisfies* $(\beta, p)$*-WWL constraint* {

*let* $\boldsymbol{q} = (0, \ldots, 0)$ *with length* $\beta - 1$*;*

/*$\boldsymbol{q}$ *is a vector storing* $\beta - 2$ *bits to the left of* $t(i)$ *in* $\boldsymbol{t}$*, with a 0 appended.*/

*if* $(i \geqslant \beta - 1)$

*let* $\boldsymbol{q}_1^{\beta-2} = \boldsymbol{t}_{i-\beta+2}^{i-1}$*;*

*else* /*$i < \beta - 1$*/

*let* $\boldsymbol{q}_{\beta-i}^{\beta-2} = \boldsymbol{t}_1^{i-1}$*;*

*find* $k \in [1 : M_{\beta-1}]$ *such that* $\boldsymbol{s}_k = \boldsymbol{q}$*.*

*let* $CntTry = cnt + \boldsymbol{X}(n - i + \beta - 1, k)$*;*

*if* $(CntTry + 1 = m)$ {

$\boldsymbol{c} = \boldsymbol{t}$*;*

*return* $\boldsymbol{c}$*; algorithm ends;*

}

*if* $(CntTry + 1 < m)$ {

*let* $\boldsymbol{c}(i) = 1$*;*

*let* $cnt = CntTry$*;*

}

}

}

**Example 2.6.2.** Suppose we would like to encode one of $M_n = 421$ $(\beta = 6, p = 3)$-WWL vectors of length $n = 10$. The message to be encoded is $m = 353$.

- $\boldsymbol{c} = (0000000000)$, $i = 1$, $\boldsymbol{t} = (1000000000)$, $\boldsymbol{q} = (00000) = s_1$, so $k = 1$. Since $cnt = 0$, $CntTry = cnt + \boldsymbol{X}(n - i + \beta - 1, k) = 236 < m - 1$, so set $cnt = 236$.

- $\boldsymbol{c} = (1000000000)$, $i = 2$, $\boldsymbol{t} = (1100000000)$, $\boldsymbol{q} = (00010) = s_3$, so $k = 3$. Compute $CntTry = cnt + \boldsymbol{X}(n - i + \beta - 1, k) = 236 + \boldsymbol{X}(13, 3) = 355 > m - 1$.

- $\boldsymbol{c} = (1000000000)$, $i = 3$, $\boldsymbol{t} = (1010000000)$, $\boldsymbol{q} = (00100) = s_5$, so $k = 5$. Compute $CntTry = cnt + \boldsymbol{X}(n - i + \beta - 1, k) = 236 + \boldsymbol{X}(12, 5) = 308 < m - 1$, so set $cnt = 308$.

- $\boldsymbol{c} = (1010000000)$, $i = 4$, $\boldsymbol{t} = (1011000000)$, $\boldsymbol{q} = (01010) = s_{11}$, so $k = 11$. Compute $CntTry = cnt + \boldsymbol{X}(n - i + \beta - 1, k) = 308 + \boldsymbol{X}(11, 11) = 343 < m - 1$, so set $cnt = 343$.

- $\boldsymbol{c} = (1011000000)$, $i = 5$, $\boldsymbol{t} = (1011100000)$ does not satisfy $(6, 3)$-WWL constraint.

- $\boldsymbol{c} = (1011000000)$, $i = 6$, $\boldsymbol{t} = (1011010000)$ does not satisfy $(6, 3)$-WWL constraint.

- $\boldsymbol{c} = (1011000000)$, $i = 7$, $\boldsymbol{t} = (1011001000)$, $\boldsymbol{q} = (11000) = s_{23}$, so $k = 23$. Compute $CntTry = cnt + \boldsymbol{X}(n - i + \beta - 1, k) = 343 + \boldsymbol{X}(8, 23) = 351 < m - 1$, so set $cnt = 351$.

- $\boldsymbol{c} = (1011001000)$, $i = 8$, $\boldsymbol{t} = (1011001100)$ does not satisfy $(6, 3)$-WWL constraint.

- $\boldsymbol{c} = (1011001000)$, $i = 9$, $\boldsymbol{t} = (1011001010)$, $\boldsymbol{q} = (00100) = s_5$, so $k = 5$. Compute $CntTry = cnt + \boldsymbol{X}(n - i + \beta - 1, k) = 351 + \boldsymbol{X}(6, 5) = 353 > m - 1$.

- $\boldsymbol{c} = (1011001000)$, $i = 10$, $\boldsymbol{t} = (1011001001)$, $\boldsymbol{q} = (01000) = s_9$, so $k = 9$. Compute $CntTry = cnt + \boldsymbol{X}(n - i + \beta - 1, k) = 351 + \boldsymbol{X}(5, 9) = 352 = m - 1$. Therefore, $\boldsymbol{c} = \boldsymbol{t} = (1011001001)$ and $ord(\boldsymbol{c}) = 353$.

**Theorem 2.6.6.** *Algorithm* 2.6.5 *encodes a message* $m \in \{1, \ldots, M_n\}$ *to a* $(\beta, p)$*-WWL vector* $\boldsymbol{c} \in \mathcal{S}_n(\beta, p)$ *such that* $ord(\boldsymbol{c}) = m$*, and its time complexity and space complexity are both* $O(n)$.

**Proof.** We first show the correctness of the algorithm and then analyze its time and space complexity.

Correctness: The proof of the correctness of the encoder is similar to the proof of the correctness of the decoder. Therefore, we omit the details.

Time complexity analysis: It can be seen from the algorithm that the encoder scans the vector from left to right once and tries to set each entry to 1. Whenever the encoder sets an entry to 1, it first determines whether the constraint is satisfied. This takes $O(1)$ steps since we

do not have to check the entire vector but only the $\beta$ bits to the left of the set entry. Then it uses binary search to find the corresponding prefix vector in $\boldsymbol{X}$, while the number of 1's is no more than $\frac{np}{\beta}$. Therefore, the complexity of the encoder is no more that $O(\frac{np}{\beta}\log M_{\beta-1}) = O(\frac{np}{\beta}\log\sum_{i=0}^{p}\binom{\beta-1}{i}) = O(n)$, where $p$ and $\beta$ are fixed numbers.

Space complexity analysis: The matrix $\boldsymbol{X}$ is the primary contributor to the space complexity. As is shown in the proof of Theorem 2.6.4, the space complexity is also $O(n)$. ■

Note that Algorithm 2.6.5 and Algorithm 2.6.3 establish a one-to-one mapping between $\{1,\ldots,M_n\}$ and $\mathcal{S}_n(\beta,p)$. Therefore the rate of the encoder is maximized. If the blocklength goes to infinity, the rate of the encoder approaches $C_{WWL}(\beta,p)$.

## 2.7 Appendix B

In this section, we present the proof of Theorem 2.4.4. The reason for which the proof of Theorem 2.4.4 is non-trivial is the following. Suppose the cell-state vector is updated from $\boldsymbol{c}_{i-1}$ to $\boldsymbol{c}_i$ on the $i$-th write. The encoder has full knowledge of $\boldsymbol{c}_{i-1}$ and $\boldsymbol{c}_i$ since we assume there is no noise in the updating procedure. The decoder is required to recover $\boldsymbol{c}_i+\boldsymbol{c}_{i-1}$ with full knowledge of $\boldsymbol{c}_i$ but zero knowledge of $\boldsymbol{c}_{i-1}$. This is similar to the situation encountered in memories with defects, considered in [37], where the most interesting scenario is when the defect locations are available to the encoder but not to the decoder. In general, this scenario can be modeled as a channel with states [22] where the side information on states is available only to the encoder.

**Proof.** First we introduce some definitions. Recall that $\mathcal{S}_n(\beta,p)$ is defined as the set all $(\beta,p)$-WWL vectors of length $n$. $\mathcal{S}_n(\beta,p)$ will be written as $\mathcal{S}$ for short if no confusion about the parameters can occur. Let $V_n=\{0,1\}^n$ be the $n$-dimensional binary vector space.

**Definition 2.7.1.** *For a vector $\boldsymbol{x}\in V_n$ and a set $\mathcal{S}\subset V_n$, we define $\mathcal{S}+\boldsymbol{x}=\{\boldsymbol{s}+\boldsymbol{x}|\boldsymbol{s}\in\mathcal{S}\}$ and denote it by $\mathcal{S}(\boldsymbol{x})$. We call vectors in $\mathcal{S}(\boldsymbol{x})$ **reachable** by $\boldsymbol{x}$ and we say $\mathcal{S}(\boldsymbol{x})$ is **centered** at $\boldsymbol{x}$.*

*For two subsets $B_1,B_2\subset V_n$, we define $B_1+B_2=\{\boldsymbol{b}_1+\boldsymbol{b}_2|\boldsymbol{b}_1\in B_1,\boldsymbol{b}_2\in B_2\}$. We call a subset $B\subset V_n$ $\mathcal{S}$**-good** if*

$$\mathcal{S}+B=\bigcup_{\boldsymbol{b}\in B}\mathcal{S}(\boldsymbol{b})=V_n,$$

*i.e., $V_n$ is covered by the union of translates of $\mathcal{S}$ centered at vectors in $B$.*

**Lemma 2.7.2.** *If $B \subset V_n$ is $\mathcal{S}$-good, then $\boldsymbol{t} + B$ is $\mathcal{S}$-good, for all $\boldsymbol{t} \in V_n$.*

**Lemma 2.7.3.** *If $B \subset V_n$ is $\mathcal{S}$-good, then for all $\boldsymbol{x} \in V_n$, there exists $\boldsymbol{b} \in B$ and $\boldsymbol{s} \in \mathcal{S}$, such that $\boldsymbol{x} + \boldsymbol{s} = \boldsymbol{b}$.*

Lemma 2.7.3 guarantees that if $B \subset V_n$ is an $\mathcal{S}$-good subset, then from any cell-state vector $\boldsymbol{x}$, there exists a $(\beta, p)$-WWL vector $\boldsymbol{s}$, such that $\boldsymbol{x} + \boldsymbol{s} \in B$. We skip the proofs of Lemma 2.7.2 and 2.7.3, referring the reader to similar results and their proofs in [15].

**Lemma 2.7.4.** *If $G_1, \ldots, G_M$ are pairwise disjoint $\mathcal{S}$-good subsets of $V_n$, then there exists a $(1, \beta, p)$-constrained code of size $M$. In particular, if $G$ is an $\mathcal{S}$-good $(n, k)$ linear code, then there exists a $(1, \beta, p)$-constrained code with rate $\frac{n-k}{n}$.*

**Proof.** If $G_i$ is $\mathcal{S}$-good for all $i \in [1 : M]$, then from Lemma 2.7.3, for any $\boldsymbol{x} \in V_n$ and $i \in [1 : M]$, there exist $\boldsymbol{g}_i \in G_i$ and $\boldsymbol{s}_i \in \mathcal{S}$, such that $\boldsymbol{x} + \boldsymbol{s}_i = \boldsymbol{g}_i$. Suppose the current cell-state vector is $\boldsymbol{x}$, then we can encode the message $i \in [1 : M]$ as a vector $\mathcal{E}(i, \boldsymbol{x}) = \boldsymbol{x} + \boldsymbol{s}_i \in G_i$, for some $\boldsymbol{s}_i \in \mathcal{S}$. The decoder uses the mapping $\mathcal{D}(\boldsymbol{x}) = i$, if $\boldsymbol{x} \in G_i$, to give an estimate of $i \in [1 : M]$. This yields a $(1, \beta, p)$-constrained code of size $M$.

If $G_1, \ldots, G_{2^{n-k}}$ represent the cosets of an $\mathcal{S}$-good $(n, k)$ linear code $G$, then each coset is $\mathcal{S}$-good according to Lemma 2.7.2. The rate of the resulting $(1, \beta, p)$-constrained code is $\frac{\log_2(2^{n-k})}{n} = \frac{n-k}{n}$. ■

Now we are ready to prove Theorem 2.4.4.

Let $B_j$ be a randomly chosen $(n, j)$ linear code with $2^j$ codewords ($B_0 = \{\boldsymbol{0}\}$), and let $m_{B_j} = |V_n \setminus (B_j + \mathcal{S})|$ be the number of vectors not reachable from any vector in $B_j$. Let $\boldsymbol{x} \in V_n$ be a randomly chosen vector and let $Q_{B_j}$ be the probability that $\boldsymbol{x} \notin B_j + \mathcal{S}$. Then we have

$$m_{B_j} = 2^n Q_{B_j}.$$

The proof of the following lemma is based upon ideas discussed in [4, pp. 201-202].

**Lemma 2.7.5.** *There exists a linear code $B_j$ such that*

$$Q_{B_j} \leqslant Q_{B_0}^{2^j}.$$

**Proof.** Let $B_j = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{2^j}\}$ denote an $(n, j)$ linear code. If

$$S_{B_j} = B_j + \mathcal{S},$$

then

$$Q_{B_j} = 1 - 2^{-n} N_{B_j},$$

where $N_{B_j} = |S_{B_j}|$.

Let $\boldsymbol{z} \notin B_j$ and let $B_{j+1,\boldsymbol{z}}$ be the $(n, j+1)$ linear code formed by $(\boldsymbol{z} + B_j) \cup B_j$. It can be seen $B_{j+1,\boldsymbol{z}}$ comprises the $2^j$ vectors in $B_j$ plus $2^j$ new vectors of the form $\boldsymbol{z} + \boldsymbol{y}$, $\boldsymbol{y} \in B_j$. Let

$$S^*_{B_j,\boldsymbol{z}} = \boldsymbol{z} + S_{B_j}$$

It can be seen that $S^*_{B_j,\boldsymbol{z}}$ has the same cardinality as $S_{B_j}$. Therefore, it contains $N_{B_j}$ vectors, too, some of which may already belong to $S_{B_j}$. Since $S_{B_{j+1,\boldsymbol{z}}} = S_{B_j} \cup S^*_{B_j,\boldsymbol{z}}$, we have

$$N_{B_{j+1,\boldsymbol{z}}} = 2N_{B_j} - \left|S_{B_j} \cap S^*_{B_j,\boldsymbol{z}}\right|.$$

Thus $N_{B_{j+1,\boldsymbol{z}}}$ is maximized by choosing $\boldsymbol{z}$ that minimizes $|S_{B_j} \cap S^*_{B_j,\boldsymbol{z}}|$.

Let us now calculate the average of $|S_{B_j} \cap S^*_{B_j,\boldsymbol{z}}|$ over all $\boldsymbol{z} \in V_n$. Here all $\boldsymbol{z} \in B_j$ are also considered since they will result in an overestimate of the average of $|S_{B_j} \cap S^*_{B_j,\boldsymbol{z}}|$. Then

$$\begin{aligned}
\sum_{\boldsymbol{z} \in V_n} |S_{B_j} \cap S^*_{B_j,\boldsymbol{z}}| &= \sum_{\boldsymbol{z} \in V_n} \sum_{\boldsymbol{x} \in S_{B_j}} \mathbf{1}_{\{\boldsymbol{x} \in S^*_{B_j,\boldsymbol{z}}\}} \\
&= \sum_{\boldsymbol{x} \in S_{B_j}} \sum_{\boldsymbol{z} \in V_n} \mathbf{1}_{\{\boldsymbol{x} \in S^*_{B_j,\boldsymbol{z}}\}} \\
&\overset{①}{=} \sum_{\boldsymbol{x} \in S_{B_j}} \sum_{\boldsymbol{z} \in \boldsymbol{x} + S_{B_j}} 1 \\
&\overset{②}{=} \sum_{\boldsymbol{x} \in S_{B_j}} N_{B_j} \\
&= N^2_{B_j},
\end{aligned}$$

where $\mathbf{1}_A$ is the indicator function of the event $A$, i.e., $\mathbf{1}_A = 1$ if $A$ is true and $\mathbf{1}_A = 0$ otherwise.

Equality ① holds since, for a fixed $\boldsymbol{x}$, if $\boldsymbol{z} \in \boldsymbol{x} + S_{B_j}$, then $\boldsymbol{x} \in S^*_{B_j,\boldsymbol{z}}$ and vice versa. Equality ② holds since $|\boldsymbol{x} + S_{B_j}| = |S_{B_j}| = N_{B_j}$. Thus, the average value of $|S_{B_j} \cap S^*_{B_j,\boldsymbol{z}}|$ is $2^{-n} N^2_{B_j}$. Since the minimum of $|S_{B_j} \cap S^*_{B_j,\boldsymbol{z}}|$ cannot exceed this average, we conclude that there exists $\boldsymbol{z} \in V_n$, such that $|S_{B_j} \cap S^*_{B_j,\boldsymbol{z}}| \leqslant 2^{-n} N^2_{B_j}$. Then there exists $B_{j+1}$, such that

$$N_{B_{j+1}} \geqslant 2N_{B_j} - 2^{-n} N^2_{B_j}.$$

Thus,

$$\begin{aligned} Q_{B_{j+1}} &= 1 - 2^{-n} N_{B_{j+1}} \\ &\leqslant 1 - 2^{-n}(2N_{B_j} - 2^{-n} N_{B_j}^2) \\ &= (1 - 2^{-n} N_{B_j})^2 \\ &= Q_{B_j}^2. \end{aligned}$$

It follows that there exists $B_j$, such that $Q_{B_j} \leqslant Q_{B_0}^{2^j}$. ■

**Lemma 2.7.6.** *If $j \geqslant n - \log|\mathcal{S}| + \log n$, then there exists $B_j$ such that $m_{B_j} < 1$.*

**Proof.** Note that $Q_{B_0} = 1 - 2^{-n} \cdot N_{B_0} = 1 - 2^{-n} \cdot |\mathcal{S}|$. Then there exists $B_j$, such that

$$\begin{aligned} Q_{B_j} &\leqslant Q_{B_0}^{2^j} \\ &\leqslant (1 - 2^{-n}|\mathcal{S}|)^{2^j} \\ &\leqslant (1 - 2^{-n}|\mathcal{S}|)^{2^{n - \log|\mathcal{S}| + \log n}} \\ &= (1 - 2^{-n}|\mathcal{S}|)^{2^n |\mathcal{S}|^{-1} \cdot n} \\ &< e^{-n} < 2^{-n}. \end{aligned}$$

Then $m_{B_j} = 2^n Q_{B_j} < 1$. ■

Since $m_{B_j}$ is an integer and $m_{B_j} < 1$, there exists an $(n, j)$ linear code $B_j$ such that $m_{B_j} = 0$, i.e., an $\mathcal{S}$-good $B_j$ exists. According to Lemma 2.7.4, there exists a sequence of $(1, \beta, p)$-constrained codes of length $n$ and rate $R_n(1, \beta, p)$ such that

$$\begin{aligned} \sup_n R_n(1, \beta, p) &\geqslant \lim_{n \to \infty} \frac{n - (n - \log|\mathcal{S}| + \log n)}{n} \\ &= \lim_{n \to \infty} \frac{\log|\mathcal{S}| - \log n}{n} \\ &= \lim_{n \to \infty} \frac{\log|\mathcal{S}|}{n} \\ &= C_{WWL}(\beta, p). \end{aligned}$$

We have seen in Theorem 2.3.2 that $C(1, \beta, p) \leqslant C_{WWL}(\beta, p)$. This concludes the proof that $C(1, \beta, p) = C_{WWL}(\beta, p)$ ■

## Acknowledgements

This chapter is in part a reprint of the material in the paper: Minghai Qin, Eitan Yaakobi, and Paul H. Siegel, “Time-space constrained codes for phase-change memories”, IEEE Transaction on Information Theory, vol. 59, no. 8, pp. 5102-5114, August 2013.

# Chapter 3

# Codes for multi-level write-once memories

## 3.1 Introduction

Chapter 2 studied time-space constraints and provides code constructions, one of which is based on binary write-once memory (WOM) codes. In this chapter, we further discuss the constructions of WOM codes with a larger alphabet size.

As part of the tremendous increase in coding research for the ubiquitous flash memories, considerable attention has been given to rewriting codes. The motivation comes from the special physical properties of the flash memory floating-gate cells, the most conspicuous of which is the asymmetric programming behavior of the cells [12]. The memory cells can only increase their level by the injection of electrons into each cell. However, in order to decrease the level of even a single cell, its entire containing block ($\sim 10^6$ cells) has to be erased. This undesired property not only reduces the writing speed but also significantly affects the lifetime of flash memories, which is often specified in terms of a maximum number of block erasures [12]. As this number can be as low as a few hundreds or thousands, reducing the number of block erasures becomes critical in improving the lifetime of flash memories.

The idea of rewriting codes dates back to the pioneering work [71] by Rivest and Shamir on write-once memory (WOM) in 1982. The motivation came from storage media such as punch cards and ablative optical disks. These media are modeled as a collection of write-once binary cells, where each cell is initially in state 0 and can be irreversibly programmed to state 1. Figure 3.1 shows a typical model for rewriting $t$ times on a binary WOM.

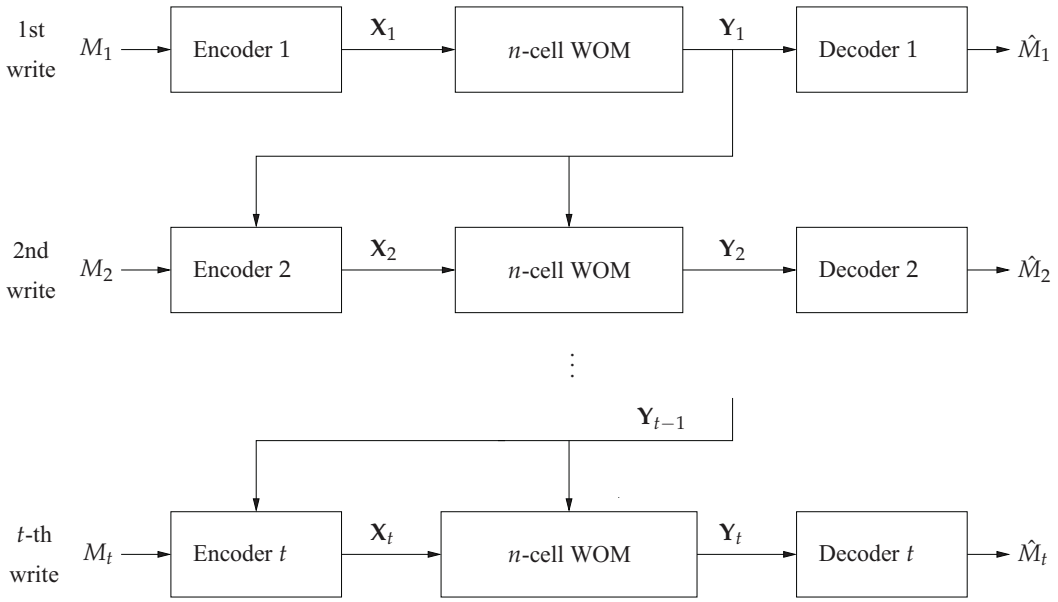


**Figure 3.1**: A $t$-write WOM model

An $[n, t; 2^{nR_{1,t}}, \ldots, 2^{nR_{t,t}}]$ WOM code consists of

- $t$ message sets $[1 : 2^{nR_{1,t}}], \ldots, [1 : 2^{nR_{t,t}}]$,
- $t$ encoders, where encoder $i \in [1 : t]$ for the $i$-th write assigns a codeword $\boldsymbol{x}_i = \mathcal{E}_i(m_i, \boldsymbol{y}_{i-1}) \in \{0,1\}^n$ ($\boldsymbol{y}_0 = \emptyset$) to each message $m_i \in [1 : 2^{nR_i}]$ and the cell levels $\boldsymbol{y}_{i-1}$ from the previous write, and
- $t$ decoders, where decoder $i \in [1 : t]$ assigns an estimate $\hat{m}_i = \mathcal{D}_i(\boldsymbol{y}_i)$ or an error $e$ to the cell levels $\boldsymbol{y}_i$ from the $i$-th write.

The notation $[i : j]$ denotes the set $\{k \in \mathbb{Z} : i \leqslant k \leqslant j\}$. The average probability of error of the WOM code is defined as $P_e^{(n)} = \mathsf{P}\{(\hat{M}_1, \ldots, \hat{M}_t) \neq (M_1, \ldots, M_t)\}$. A rate tuple $(R_{1,t}, \ldots, R_{t,t})$ is said to be achievable for the WOM if there exists a sequence of $[n, t; 2^{nR_{1,t}}, \ldots, 2^{nR_{t,t}}]$ WOM codes such that $\lim_{n\to\infty} P_e^{(n)} = 0$. The capacity region $\mathscr{C}_{\text{WOM}}(t)$ is the closure of the set of all achievable rate tuples $(R_{1,t}, \ldots, R_{t,t})$. The sum-capacity $C_{\text{sum}}(t)$ of WOM is the maximum achievable sum-rate $\sum_{j=1}^{t} R_{j,t}$. A sequence of WOM codes is said to be sum-rate optimal if its sum-rate approaches the sum-capacity in the limit.

The capacity region as well as the sum-capacity for the WOM model is well studied in the literature [28, 36]; for example, it is known that the $t$-write sum-capacity for binary WOM is

$C_{\text{sum}}(t) = \log_2(t+1)$ and the capacity region is

$$\mathscr{C}_{\text{WOM}}(t) = \Big\{(R_{1,t}, \ldots, R_{t,t}) | R_{1,t} \leqslant H(p_1),$$
$$R_{i,t} \leqslant \left(\prod_{k=1}^{i-1} p_k\right) H(p_i),\ i \in [2 : t-1],$$
$$R_{t,t} \leqslant \prod_{k=1}^{t-1} p_k \text{ for some } p_1, \ldots, p_{t-1} \in [\tfrac{1}{2}, 1]\Big\}.$$

Considerable progress has been made in the last few years in the study of binary WOM code constructions for single-level cell (SLC) flash memory devices where each cell supports $q = 2$ levels [43,93,95]. However, to increase storage densities, future flash memory devices are expected to support a large number of cell levels, continuing the trend seen with the use of MLC and TLC devices that support 4 and 8 levels, respectively. This has motivated a body of work on the construction of WOM codes for multilevel cells that support $q \geqslant 3$ levels [13,31,53,69,76]. The sum-capacity of rewrite codes for $t$ writes on $q$-ary cells is known to be $\log_2 \binom{q+t-1}{q-1}$ [28], although explicit characterization of the capacity region remains an open problem.

In this chapter, we consider the construction of lattice-based WOM codes for $t$ writes on $q$-level cells. Lattice-based 2-write WOM codes over $n$ cells in the asymptotic setting of continuous cell levels were derived in [53] for the fixed-rate scenario, where the cardinality of the message set is the same on each write. Allowing the cardinality of the message set on each write to be different can increase the sum-rate. Using a continuous approximation approach, it was hypothesized in [53] that the hyperbolic shaping regions were optimal for maximizing sum-rates of two writes over lattices in arbitrary dimensions. Optimality of hyperbolic shaping regions was proven in [6] for lattices in $n = 2$ dimensions when the number of writes, $t$, is arbitrary. A proof of optimality was provided in [54] for the case of an arbitrary number of cells, $n$, and $t = 2$ writes.

Here, we consider the most general case of an arbitrary number of writes on an arbitrary number of cells where each cell supports a large number of levels. Using the continuous approximation approach we prove that hyperbolic shaping regions are optimal for maximizing the sum-rate. The results are then extended to the fixed-rate case, a scenario of practical importance.

The rest of the chapter is organized as follows. Section 3.2 formulates the code design problem. In Section 3.2.3, we extend ideas presented in [54] and invoke the continuous approximation to obtain an upper bound on the worst-case sum-rate optimal $t$-write regions for $n$ cells and consider its asymptotic behavior as the number of cells grows large. In Section 3.3, we derive the worst-case fixed-rate optimal $t$-write regions for $n$ cells.

## 3.2 Lattice-based WOM Codes

### 3.2.1 Lattices and Lattice Codes

**Definition 3.2.1.**

*An $n$-dimensional lattice $\Lambda$ is defined by a generator matrix $G \in \mathbb{R}^{n\times n}$ and we denote the lattice by $\Lambda(G)$ or simply $\Lambda$ is no confusion can occur. Let $\boldsymbol{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ and $\boldsymbol{b} = (b_1, b_2, \ldots, b_n) \in \mathbb{Z}^n$ be $n$-dimensional real-valued vector and integer-valued vector, respectively, then the lattice $\Lambda(G)$ is defined as*

$$\Lambda = \{\boldsymbol{x} \in \mathbb{R}^n | \exists \boldsymbol{b} \in \mathbb{Z}^n, \boldsymbol{x} = \boldsymbol{b} \cdot G, \}.$$

*The Voronoi region of a point $\boldsymbol{x} \in \Lambda$ is the set of points of $\mathbb{R}^n$ which are closer to $\boldsymbol{x}$ than to any other point $\boldsymbol{x}' \in \Lambda$, i.e.,*

$$\textit{Voronoi}(\boldsymbol{x}) = \{\boldsymbol{y} \in \mathbb{R}^n | d(\boldsymbol{y}, \boldsymbol{x}) \leqslant d(\boldsymbol{y}, \boldsymbol{z}), \forall \boldsymbol{z} \in \Lambda\},$$

*where $d(\cdot, \cdot)$ is the Euclidean distance between two points.*

*The volume of a Voronoi region* $\mathrm{Vol}(\Lambda)$ *is*

$$\mathrm{Vol}(\Lambda) \triangleq |\det \mathrm{G}|. \tag{3.1}$$

*A lattice code $\mathcal{L}$ is a finite subset of a lattice $\Lambda$, described by a shaping region $\mathbb{A} \subset \mathbb{R}^n$,*

$$\mathcal{L} = \Lambda \cap \mathbb{A}. \tag{3.2}$$

For a thorough treatment of lattices, refer to [17].

The coordinate values of an $n$-dimensional lattice code may be stored in $n$ cells of a flash memory. In the most general case, cell $j$ stores a continuous value between 0 and $\ell_j$, so that the stored values in $n$ cells are represented by $\boldsymbol{x} \in [0, \ell_1] \times [0, \ell_2] \times \ldots [0, \ell_n] \triangleq \mathbb{A}$ where $x_j \in \mathbb{R}, \forall\, j = 1, 2, \ldots, n$. The volume of region $\mathbb{A}$ is $\|\mathbb{A}\| = \prod_{j=1}^{n} \ell_j$. Allowing arbitrary $\ell_j$ proves to be helpful in the sequel. However, in the typical case of a $q$-ary flash memory, we have $\ell_j = q - 1$ for all $j$, so $\mathbb{A} = [0, q-1]^n$ and the codebook is $\mathcal{L} = \mathbb{Z}_q^n$.

### 3.2.2 WOM Codebooks

Suppose the number of flash memory cells is $n$ and they are represented by the vector $(x_1, x_2, \ldots, x_n) \in \mathcal{L}$ such that the level of a cell can only be increased during a write operation. We consider updating information in these $n$ cells $t$ times before a block erasure is required.

**Definition 3.2.2.** *A* $t$-write WOM code *stores* $M_i$ *messages in* $n$ *cells in the worst case at write* $i$, $i = 1, \ldots, t$. *The instantaneous rate for write* $i$ *and the* worst-case sum-rate *for the* $t$*-write code are*

$$R_{i,t} = \frac{1}{n} \log_2 M_i \text{ bits per cell per write, and} \tag{3.3}$$

$$R_t = \sum_{i=1}^{t} R_{i,t} \text{ bits per cell per erase, respectively.} \tag{3.4}$$

*A lattice-based* $t$*-write WOM codebook is defined by a partition of a lattice code* $\mathcal{L}$ *into* $t$ *disjoint subsets, denoted as* $\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_t$. *The subset* $\mathcal{L}_i$ *is the codebook for the* $i$*-th write and has cardinality* $|\mathcal{L}_i|$.

Note that since these codes have disjoint codebooks for each write, they are a special case of WOM codes referred to as synchronized WOM codes in [71].

A point $\boldsymbol{x} \in \mathcal{L}$ is said to be accessible from another point $\boldsymbol{s}$, denoted as $\boldsymbol{x} \succ \boldsymbol{s}$, if $x_j \geqslant s_j, \forall j in [1:n]$ and $\boldsymbol{s} \neq \boldsymbol{x}$. Suppose the point stored at the $(i-1)$-st write is $\boldsymbol{s}$, then the set of points that may be stored at the $i$-th write is

$$\mathcal{L}_i(\boldsymbol{s}) \triangleq \{\boldsymbol{x} \in \mathcal{L}_i : \boldsymbol{x} \succ \boldsymbol{s}\}. \tag{3.5}$$

Here, $\mathcal{L}_i(\mathbf{s})$, the subset of $\mathcal{L}_i$ accessible from $\mathbf{s}$, may be a proper subset of the codebook $\mathcal{L}_i$. Since the worst-case rate is of interest, define codebook cardinality, denoted by $C_i$, as the minimum number of points in $\mathcal{L}_i$ that are accessible from any point in $\mathcal{L}_{i-1}$,

$$C_i \triangleq \min_{\boldsymbol{s} \in \mathcal{L}_{i-1}} |\mathcal{L}_i(\mathbf{s})|. \tag{3.6}$$

Also define the total codebook cardinality, $\Pi_t$, as

$$\Pi_t \triangleq \prod_{i=1}^{t} C_i. \tag{3.7}$$

The state of the memory before the first write is $\boldsymbol{s} = \mathbf{0}$ and all points in the codebook $\mathcal{L}_1$ are accessible. Thus, $M_1 = C_1$. However, the set of points that may be stored at any $i$-th write, $i > 1$, depends on the point stored on the previous write. As a result, there may not exist a scheme which can consistently map $C_i$ messages to points in the codebook $\mathcal{L}_i$. In some cases, then, $M_i$ may be smaller than $C_i$,

$$M_i \leqslant C_i, \tag{3.8}$$

and accordingly each rate $R_{i,t}$ is upper bounded as,

$$R_{i,t} \leqslant \frac{1}{n} \log_2 C_i, \tag{3.9}$$

and the worst-case sum-rate is upper bounded as

$$R_t \leqslant \frac{1}{n} \log_2 \Pi_t. \tag{3.10}$$

In this chapter, we concentrate on maximizing $C_i$ and $\Pi_t$ because they provide upper bounds on $R_{i,t}$ and $R_t$, respectively. The matter of consistent encoding-decoding is discussed in [7].

### 3.2.3 Continuous Approximation

According to the continuous approximation principle for dense lattices [24,25], the number of points in a codebook $\mathcal{L}$ formed using (3.2) can be approximated as

$$|\mathcal{L}| \approx \frac{\|\mathbb{A}\|}{\mathrm{Vol}\,(\Lambda)}, \tag{3.11}$$

where $|\mathcal{L}|$ denotes the cardinality of the discrete set $\mathcal{L}$ and $\|\mathbb{A}\|$ denotes the volume of the shaping region $\mathbb{A}$. This approximation becomes increasingly accurate as the density of the lattice increases. The use of the continuous approximation principle for WOM codes was introduced in [54].

WOM codebooks $\mathcal{L}_1, \ldots, \mathcal{L}_t$ may be constructed by partitioning $\mathbb{A}$ into $t$ write regions, $\mathbb{A}_1, \mathbb{A}_2, \ldots, \mathbb{A}_t$. To construct codebooks for cells that support discrete levels, let

$$\mathcal{L}_i = \mathcal{L} \cap \mathbb{A}_i. \tag{3.12}$$

Applying the continuous approximation to the individual write regions, the codebook cardinality for the first write is approximated by

$$C_1 = |\mathcal{L}_1| \approx \frac{\|\mathbb{A}_1\|}{\mathrm{Vol}\,(\Lambda)} \triangleq V_1. \tag{3.13}$$

If the state of the memory after the $(i-1)$-st write is $\mathbf{s} \in \mathbb{A}_{i-1}$, then the set of possible levels that can be written on the $i$-th write is

$$\mathbb{A}_i(\boldsymbol{s}) \triangleq \{\boldsymbol{x} \in \mathbb{A}_i : \mathbf{x} \succ \boldsymbol{s}\}. \tag{3.14}$$

Applying the continuous approximation for writes $2, 3, \ldots, t$, the codebook cardinality is approximated by

$$C_i \approx \frac{1}{\mathrm{Vol}(\Lambda)} \inf_{\mathbf{s} \in \mathbb{A}_{i-1}} \|\mathbb{A}_i(\boldsymbol{s})\| \triangleq V_i, \tag{3.15}$$

and the total codebook cardinality in $t$ writes is approximated by

$$\Pi_t \approx \prod_{i=1}^{t} V_i \triangleq S_t. \tag{3.16}$$

In the following sections, the quantities $V_i$ and $S_t$ are also referred to as the codebook cardinality and the total codebook cardinality, respectively. Since both the lattice and the maximum cell values $\ell_i$ may be scaled arbitrarily, in Section 3.3 we assume that $\mathrm{Vol}(\Lambda) = 1$.

## 3.3 Fixed-Rate Optimal $t$-Writes

In this section, we do not design WOM codebooks directly. Rather, we select shaping regions $\mathbb{A}_1, \mathbb{A}_2, \ldots, \mathbb{A}_t$ and maximize the total codebook cardinality $S_t$. Under the continuous approximation principle, this corresponds to maximizing the upper bound on the worst-case sum rate (3.10).

In practice, it might be preferable to constrain successive writes to have the same rate. Therefore, we consider shaping regions $\mathbb{A}_1, \ldots, \mathbb{A}_t$ such that the codebook cardinality is constant, i.e., $V_1 = V_2 = \cdots = V_t$. The problem of maximizing the sum-rate without the constraint of equal rates is studied in [7].

We start with the following definition.

**Definition 3.3.1.** *Let $\mathbb{H}(u)$ denote the region in $\mathbb{A}$ enclosed by an $n$-dimensional rectangular hyperbola with parameter $u$, i.e.,*

$$\mathbb{H}(u) \triangleq \left\{ \mathbf{x} \in \mathbb{A} : \prod_{j=1}^{n} (\ell_j - x_j) \geqslant u \cdot \|\mathbb{A}\| \right\}, \tag{3.17}$$

*with $0 \leqslant u \leqslant 1$. The* **normalized volume** *of the region $\mathbb{H}(u)$ is denoted as*

$$\Delta(u) \triangleq \|\mathbb{A}\|^{-1} \cdot \|\mathbb{H}(u)\|. \tag{3.18}$$

Figure 3.2 shows a 2-dimensional rectangular hyperbola $\mathbb{H}(u)$.

Note that the parameter $u$, given in Definition 3.3.1, characterizes the point where the hyperbola touches the axes, and is also equal to the normalized volume of the region accessible from any point on the boundary of the hyperbola. $\Delta(u)$ is equal to the normalized volume of the region under the hyperbola. and it can be expressed in closed form as follows.

**Lemma 3.3.2.** *For $n \geqslant 2$,*

$$\Delta(u) = 1 - u \sum_{i=0}^{n-1} \frac{1}{i!} \left[ \ln\left(\frac{1}{u}\right) \right]^i \tag{3.19}$$

**Proof.** See [7] for details. ■

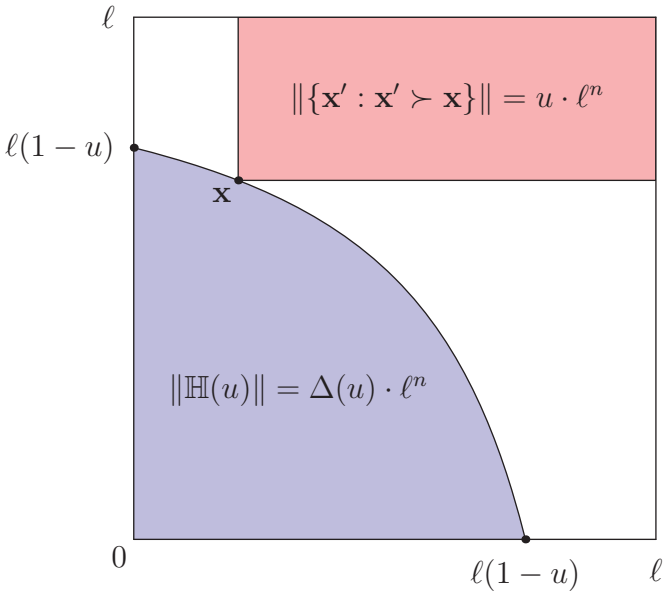


**Figure 3.2**: A 2-dimensional rectangular hyperbola $\mathbb{H}(u)$ in region $\mathbb{A} = [0, \ell] \times [0, \ell]$. The region under the hyperbola and the region accessible from a given point $\mathbf{x}$ on the hyperbola are shaded in blue and red, respectively, and their volumes are equal to $\Delta(u) \cdot \ell^n$ and $u \cdot \ell^n$, respectively.

The following theorem gives the optimal write-regions under the constraint that each write has equal rate.

**Theorem 3.3.3 (Fixed-rate optimal $t$-writes)** *The unique optimal boundary for write $i$ when storing information $t$ times on $n$ cells such that the total codebook cardinality is maximized and the codebook cardinality on each write is the same is given by*

$$\mathbb{B}_i^* = \left\{ \boldsymbol{x} \in \mathbb{A} : \prod_{j=1}^{n} (\ell_j - x_j) = \prod_{m=t-i+1}^{t} v_m^* \cdot \|\mathbb{A}\| \right\} \tag{3.20}$$

*for all $i = 1, \ldots, t-1$ where $v_1^* = 0$, and for $k \geqslant 2$, $v_k^*$ satisfies*

$$\Delta\left(v_k^*\right) = v_k^* \cdot \Delta\left(v_{k-1}^*\right). \tag{3.21}$$

*The codebook cardinality on write $i$, $i = 1, \ldots, t$, is*

$$V_{fix}^* = \Delta\left(v_t^*\right) \cdot \|\mathbb{A}\|, \tag{3.22}$$

*and the total codebook cardinality in $t$ writes is*

$$S_{fix}^*(\ell_1, \ldots, \ell_n) = \left(\Delta\left(v_t^*\right) \cdot \|\mathbb{A}\|\right)^t. \tag{3.23}$$

**Proof.** We prove the theorem by induction. First, suppose $t = 2$. Let the boundary be

$$\mathbb{B}' = \left\{ \boldsymbol{x} \in \mathbb{A} \,:\, \prod_{j=1}^{n} (\ell_j - x_j) = v_2^* \cdot \|\mathbb{A}\| \right\} \tag{3.24}$$

so that the first write region $\mathbb{A}_1(\mathbb{B}')$ is $\mathbb{H}(v_2^*)$. The second write region $\mathbb{A}_2$ is $\mathbb{A} \setminus \mathbb{A}_1(\mathbb{B}')$ and $\mathbb{A}_2(\boldsymbol{x}_1) = \{\boldsymbol{x} \in \mathbb{A}_2 : \boldsymbol{x} \succ \boldsymbol{x}_1\}$ for all $\boldsymbol{x}_1 \in \mathbb{A}_1(\mathbb{B}')$. Then, the codebook cardinality in each of the two writes, under the constraint that the rates are equal, is

$$\begin{aligned} V_{\text{fix}}(\mathbb{B}') &= \min \left\{ \left\|\mathbb{A}_1(\mathbb{B}')\right\| \,,\, \min_{\boldsymbol{x}_1 \in \mathbb{A}_1} \|\mathbb{A}_2(\boldsymbol{x}_1)\| \right\} && (3.25)\\ &= \min \left\{ \left\|\mathbb{A}_1(\mathbb{B}')\right\| \,,\, \min_{\boldsymbol{x} \in \mathbb{B}'} \|\mathbb{A}_2(\boldsymbol{x})\| \right\} && (3.26)\\ &= \min \left\{ \Delta(v_2^*) \cdot \|\mathbb{A}\| \,,\, v_2^* \cdot \|\mathbb{A}\| \right\} && (3.27)\\ &= \Delta(v_2^*) \cdot \|\mathbb{A}\| \;=\; v_2^* \cdot \|\mathbb{A}\| && (3.28) \end{aligned}$$

where (3.26) follows from the fact that the minimum is achieved when $\boldsymbol{x}_1$ lies on the boundary $\mathbb{B}'$ and (3.28) follows from (3.21). Now, suppose the optimal write regions are $\mathbb{A}_1^*$ and $\mathbb{A}_2^*$ with maximum codebook cardinality $V_{\text{fix}}^*$. Then $V_{\text{fix}}(\mathbb{B}') \leqslant V_{\text{fix}}^*$. We will prove that $\mathbb{A}_1^* \subseteq \mathbb{A}_1(\mathbb{B}')$. Suppose the opposite is true; that is, suppose there exists $\boldsymbol{x}' \in \mathbb{A}_1^*$ such that $\prod_j (\ell_j - x_j') < v_2^* \cdot \|\mathbb{A}\|$. Since the codebook cardinality $V_{\text{fix}}^*$ is upper bounded by the volume of the second region,

$$V_{\text{fix}}^* \leqslant \left\|\mathbb{A}_2^*(\boldsymbol{x}')\right\| = \prod_j (\ell_j - x_j') < v_2^* \cdot \|\mathbb{A}\| = V_{\text{fix}}(\mathbb{B}'), \tag{3.29}$$

which contradicts the optimality of $\mathbb{A}_1^*$. Thus, the optimal first-write region $\mathbb{A}_1^*$ must be contained in $\mathbb{A}_1(\mathbb{B}')$. On the other hand, by the optimality of $\mathbb{A}_1^*$ and $\mathbb{A}_2^*$,

$$\begin{aligned} V_{\text{fix}}(\mathbb{B}') \leqslant V_{\text{fix}}^* &\leqslant \|\mathbb{A}_1^*\| && (3.30)\\ &\leqslant \left\|\mathbb{A}_1(\mathbb{B}')\right\| = V_{\text{fix}}(\mathbb{B}'), && (3.31) \end{aligned}$$

which implies that

$$V_{\text{fix}}^* = \|\mathbb{A}_1^*\| = V_{\text{fix}}(\mathbb{B}') = \Delta(v_2^*) \cdot \mathbb{A}. \tag{3.32}$$

To sum up, $\mathbb{A}_1^* \subseteq \mathbb{A}_1(\mathbb{B}')$ and $\|\mathbb{A}_1^*\| = \|\mathbb{A}_1(\mathbb{B}')\|$. It now follows from [7, Lemma A.1] that $\mathbb{A}_1^* = \mathbb{A}_1(\mathbb{B}')$. It can similarly be shown that $\mathbb{A}_2^* = \mathbb{A} \setminus \mathbb{A}_1(\mathbb{B}')$. This proves that the theorem holds for $t = 2$.

Now suppose the theorem holds when the number of writes is $k$. Consider the case for $k+1$ writes. Suppose $v_{k+1}^*$ satisfies $\Delta(v_{k+1}^*) = v_{k+1}^* \cdot \Delta(v_k^*)$. Define the boundary for the first-write region as

$$\mathbb{B}'_{1,k+1} = \left\{ \boldsymbol{x} \in \mathbb{A} : \prod_{j=1}^{n} (\ell_j - x_j) = v_{k+1}^* \cdot \|\mathbb{A}\| \right\} \tag{3.33}$$

so that the first write region is $\mathbb{A}_{1,k+1} = \mathbb{H}\left(v_{k+1}^*\right)$. Let $V_{\text{fix},k}^*(\ell_1 - x_1, \ldots, \ell_n - x_n)$ denote the optimal codebook cardinality for the last $k$ writes in the region $[x_1, \ell_1] \times \ldots \times [x_n, \ell_n]$. Then the codebook cardinality on each subsequent write is given by

$$\begin{aligned}
V_{\text{fix},k+1} &\leqslant \min\left\{ \|\mathbb{A}_{1,k+1}\| \, , \min_{\boldsymbol{x} \in \mathbb{A}_{1,k+1}} V_{\text{fix},k}^*(\ell_1 - x_1, \ldots, \ell_n - x_n) \right\} && (3.34)\\
&= \min\left\{ \left\|\mathbb{H}(v_{k+1}^*)\right\| \, , \min_{\boldsymbol{x} \in \mathbb{A}_{1,k+1}} \Delta(v_k^*) \cdot \prod_{j=1}^{n} (\ell_j - x_j) \right\} && (3.35)\\
&= \min\left\{ \left\|\mathbb{H}(v_{k+1}^*)\right\| \, , \min_{\boldsymbol{x} \in \mathbb{B}'_{1,k+1}} \Delta(v_k^*) \cdot \prod_{j=1}^{n} (\ell_j - x_j) \right\} && (3.36)\\
&= \min\left\{ \Delta(v_{k+1}^*) \cdot \|\mathbb{A}\| \, , \Delta(v_k^*) \cdot v_{k+1}^* \cdot \|\mathbb{A}\| \right\} && (3.37)\\
&= \Delta(v_{k+1}^*) \cdot \|\mathbb{A}\| = \Delta(v_k^*) \cdot v_{k+1}^* \cdot \|\mathbb{A}\| \, . && (3.38)
\end{aligned}$$

Equality (3.35) follows from the induction hypothesis, (3.36) follows from the fact that the minimum occurs when $\boldsymbol{x}$ lies on $\mathbb{B}'_{1,k+1}$, and (3.38) follows from (3.21). In a manner similar to the proof of [7, Theorem III.4], it can be shown that the optimal boundary for the $i^{\text{th}}$ subsequent write after storing a point $\boldsymbol{x}' \in \mathbb{B}'_{1,k+1}$ on the first write is given by

$$\begin{aligned}
\mathbb{B}_{i+1,k+1}^* &= \boldsymbol{x}' + \mathbb{B}_{i,k}^*\left(\ell_1 - x_1', \ldots, \ell_n - x_n'\right) && (3.39)\\
&= \left\{ \boldsymbol{x} \in \mathbb{A} : \prod_{j=1}^{n} (\ell_j - x_j) = \prod_{m=k-i+1}^{k} v_m^* \cdot v_{k+1}^* \cdot \|\mathbb{A}\| \right\} && (3.40)\\
&= \left\{ \boldsymbol{x} \in \mathbb{A} : \prod_{j=1}^{n} (\ell_j - x_j) = \prod_{m=k-i+1}^{k+1} v_m^* \cdot \|\mathbb{A}\| \right\} . && (3.41)
\end{aligned}$$

From (3.41), $\mathbb{B}_{i+1,k+1}^*$ is independent of $\boldsymbol{x}'$, the point stored on the first write, and the inequality in (3.34) is in fact an equality. Thus, the claim in the theorem is true for $k+1$ writes. This proves the theorem by induction. ■

### 3.3.1 Computing the optimal hyperbola parameters, $v_t^*$

The hyperbola parameters for optimal fixed-rate write-regions can be computed easily for the case of $n = 2$ cells.

**Proposition 3.3.4** *For $n = 2$ cells, the optimal hyperbola parameters, $v_k^*$, are given by the recurrence relation*

$$v_k^* = \frac{-1}{W_{-1}\left(-\exp\left(-1 - \prod_{m=1}^{k-1} v_m^*\right)\right)} \tag{3.42}$$

**Proof.** For convenience of notation, we drop the asterisk from $v_k^*$. For $k \geqslant 2$, the optimal hyperbola parameter, $v_k$, satisfies

$$\Delta(v_k) = v_k \cdot \Delta(v_{k-1}) = v_k \cdot \Theta_k$$

where $\Theta_k \triangleq \prod_{m=1}^{k-1} v_m$. Using Lemma 3.3.2 for $n = 2$, we get

$$\frac{-1}{v_k} \cdot \exp\left(\frac{-1}{v_k}\right) = -e^{-(1+\Theta_k)} \triangleq \xi_k.$$

By the definition of Lambert W function, $-v_k^{-1} = W(\xi_k)$. However, $W_0(\xi_k) \neq -{v_k}^{-1}$ since $W_0(x) > -1$ for all $x$. By induction on $k$, $\xi_k$ lies in the domain of $W_{-1}$. Therefore $v_k = -\left(W_{-1}(\xi_k)\right)^{-1} \in (0, 1]$. ■

## Acknowledgments

This chapter is in part a reprint of the material in the paper: Aman Bhatia, Minghai Qin, Aravind Iyengar, Brian Kurkoski, and Paul H. Siegel, "Lattice-based WOM codes for multilevel flash memories", IEEE Journal on Selected Areas in Communications, vol. 32, no. 5, pp. 933-945, May 2014.

# Chapter 4

# Write-once memories with retained messages

## 4.1 Introduction and Main Results

In Chapter 3, we gave explicit WOM code constructions for the canonical WOM model. In this chapter, we continue the discussion of WOM model and extend it to some practical scenarios.

Following the work by Rivest and Shamir on the binary WOM, many papers on WOM codes appeared during the 1980s and 1990s, (e.g., [27,28,36,91]) as well as in the past few years (e.g., [30,43,51,88,93]). Among all of the existing models for rewriting on flash memories, one assumes that a new message is stored in the memory on each write, effectively overwriting previously written messages. This can be a drawback in some applications, however, if the user wishes to retain access to one or more previously written messages. For example, suppose that a police station keeps traffic surveillance videos for up to a certain amount of time, say 30 days. This requires that the most current video as well as the videos from the previous 29 days be retrievable at any time. If the entire set of 30 daily videos are treated as a completely new message to be written on top of the existing content of the memory cells, the writing efficiency will be low, because the same message is being written multiple times via different codewords.

This motivates the model of rewriting flash memories with retained messages. This model is related to the work on buffer codes and trajectory codes [43, 45], which are capable of remembering the most recent values stored in the memories. To make the problem simple, in this chapter we consider rewriting on a binary WOM, where after each write the current message

and some of the previously written messages need to be retrievable. We aim to characterize the optimal rate trade-off and find code constructions focusing on three concrete problems motivated by different scenarios in real storage systems.

In Section 4.2, we formulate the problem of retaining two days of video surveillance as follows.

**Problem 1. Consecutive two-step WOM**
On the $i$-th write, $i \in [1:t]$, encoder $i$ stores $(M_{i-1}, M_i)$ ($M_0 = \emptyset$) and decoder $i$ has to recover both messages.

**Remark 4.1.1.** In this chapter, the notations are inherited from Chapter 3. Since we assume that the number of writes is fixed to be $t$, for simplicity we will denote $R_{i,t}$, the rate on the $i$-th write for $i \in [1:t]$, by $R_i$ if no confusion could occur.

By ignoring the correlation between message pairs over multiple writes and treating $(M_{i-1}, M_i)$ as a new message, one can achieve roughly $\frac{1}{2}\log_2(t+1)$ in sum-rate using a traditional WOM code, since every $M_i, i \in [1:t-1]$, is written twice. Is this optimal? We establish in Theorem 4.2.3 that the sum-capacity of the consecutive two-step WOM model is $\log_2\left(\lceil \frac{t}{2} \rceil + 1\right)$, which can be twice as large as $\frac{1}{2}\log_2(t+1)$ for large $t$. How can we fully exploit the correlation among messages? We propose in Construction 4.2.1 a very simple code, which turns out to be sum-rate optimal. The idea is to partition the set of $n$ cells into two blocks and to update the new message alternately on the two blocks, as shown in Table 4.1. An outer bound on the capacity region for general $t$ is also derived.

**Table 4.1**: Writing arrangement of the consecutive 2-step WOM code

| | block 1 | block 2 |
|---|---|---|
| 1st write | $M_1$ | |
| 2nd write | $M_1$ | $M_2$ |
| 3rd write | $M_3$ | $M_2$ |
| 4th write | $M_3$ | $M_4$ |
| 5th write | $M_5$ | $M_4$ |

Now suppose the police station wishes to keep track of the video from the most recent day on which there was a traffic accident, as well as the traffic video for the current day. Since the traffic accident is unpredictable, we cannot tell which part of the video will be kept for the next day until the end of the current day. This is the situation in which the retrievable message for the next write can be an arbitrary one from the two messages currently written in the memory. To be more concrete, we formulate the problem as follows.

**Problem 2. Arbitrary two-step WOM**

On the first write, encoder 1 stores message $M_1$ and decoder 1 has to recover $M_1$. On the $i$-th write, $i \in [2 : t]$, encoder $i$ stores $(M_{s(i)}, M_i)$, where $M_{s(i)} \in \{M_{s(i-1)}, M_{i-1}\}$ is arbitrarily chosen from the two messages stored on the $(i-1)$-st write, and decoder $i$ has to recover both messages.

For this problem, an idea arises naturally from the construction for the consecutive two-step WOM. With Table 4.1 in mind, we store $M_1$ and $M_2$ the same way as before for the first two writes. If $(M_1, M_3)$ is stored on the third write, we update $M_3$ on block 2. If instead $(M_2, M_3)$ is stored on the third write, we update $M_3$ on block 1. It can be shown that the sum-rate is roughly $\frac{1}{2}\log_2(t)$ in the worst case scenario. Can we do better than this? In Section 4.3, we construct a code by enlarging the number of blocks, and we show that it can strictly outperform the above code. Moreover, it is shown to be asymptotically optimal in $t$. A simple outer bound on the capacity region and an upper bound on the sum-capacity are also presented.

Now we introduce the last problem. Suppose the surveillance videos are layered as high-fidelity and low-fidelity ones, e.g., encoded by the H.264 standard. On each day, all low-fidelity videos from previous days and the high-fidelity video from the current day should be stored. This motivates the following.

**Problem 3. Incremental WOM**

We rewrite each message $M_i$, $i \in [1 : t]$, as two independent parts: the common message $M_i^c \in [1 : 2^{nR_i^c}]$ and the private message $M_i^p \in [1 : 2^{nR_i^p}]$, i.e., $M_i = (M_i^c, M_i^p)$. On the $i$-th write, encoder $i$ stores all the previous common messages and its own full message, i.e., $(M_1^c, M_2^c, \ldots, M_i^c, M_i^p)$, and decoder $i$ has to recover all of them.

One extreme special case of this problem is $M_i^c = \emptyset, i \in [1 : t]$, i.e., there is no common message. Then we go back to the traditional $t$-write WOM. The other extreme special case is $M_i^p = \emptyset, i \in [1 : t]$, i.e., there is no private message. Since all the previously written messages have to be recoverable by the current decoder, the performance is fundamentally limited by the last write. It can be shown that the capacity region for this extreme problem is $\sum_{i=1}^t R_i \leqslant 1$. Thus, an obvious choice to maximize the sum-rate is to set all the common-message rates to be zero and the sum-capacity is readily established as $\log_2(t+1)$. In Section 4.4, we establish the optimal trade-off between the common-message sum-rate $R_{\text{sum}}^c := \sum_{i=1}^n R_i^c$ and the private-message sum-rate $R_{\text{sum}}^p := \sum_{i=1}^n R_i^p$.

Moreover, we investigate the symmetric sum-capacity $C_{\text{ssum}}^3(t)$, defined as the maximum

achievable sum-rate when $R_1^c = R_1^p = R_2^c = R_2^p = \cdots = R_t^c = R_t^p = R$. Since the problem formulation is apparently a combination of two completely solved extreme problems, one might think that a time-sharing strategy between the two optimal coding schemes would be optimal. Surprisingly, in Section 4.4, we construct a code that strictly outperforms the time-sharing code and is asymptotically optimal in $t$. The performance of this construction is illustrated in Figure 4.2.

## 4.2 Consecutive Two-Step WOM

In this section we establish the sum-capacity as well as an outer bound and an inner bound on the capacity region for the consecutive two-step WOM defined in Problem 1.

**Proposition 4.2.1.** *(Outer bound on the capacity region) If a rate tuple $(R_1, R_2, \ldots, R_t)$ is achievable for the $t$-write consecutive two-step WOM, it must satisfy $R_1 \leqslant H(Y_1), R_1 + R_2 \leqslant H(Y_2), R_2 + R_3 \leqslant H(Y_3|Y_1), R_3 + R_4 \leqslant H(Y_4|Y_2), \ldots, R_{t-1} + R_t \leqslant H(Y_t|Y_{t-2})$ for some pmf $p(x_1)p(x_2|y_1)\cdots\ p(x_t|y_{t-1})$.*

**Proof.** See Appendix. ■

**Proposition 4.2.2.** *(Inner bound on the capacity region) For even $t$, let $s = t/2$. If two rate tuples $(R'_1, \ldots, R'_s)$ and $(R''_1, \ldots, R''_s)$ are achievable for the $s$-write WOM, then for all $\lambda \in [0, 1]$, with $\breve{} = 1 - \lambda$, the rate tuple $(R_1, \ldots, R_t) = (\lambda R'_1, \breve{} R''_1, \lambda R'_2, \breve{} R''_2, \ldots, \lambda R'_s, \breve{} R''_s)$ is achievable for the $t$-write consecutive two-step WOM.*

*For odd $t$, let $s = (t-1)/2$. If the rate tuple $(R'_1, \ldots, R'_{s+1})$ is achievable for the $(s+1)$-write WOM and the rate tuple $(R''_1, \ldots, R''_s)$ is achievable for the $s$-write WOM, then for all $\lambda \in [0, 1]$, the rate tuple $(R_1, \ldots, R_t) = (\lambda R'_1, \breve{} R''_1, \lambda R'_2, \breve{} R''_2, \ldots, \lambda R'_s, \breve{} R''_s, \lambda R'_{s+1})$ is achievable for the $t$-write consecutive two-step WOM.*

**Proof.** See Appendix. ■

The above outer and inner bounds coincide at the sum-rate and establish the sum-capacity of the consecutive two-step WOM for every $t$.

**Theorem 4.2.3.** *The sum-capacity $C^1_{\text{sum}}(t)$ of the $t$-write consecutive two-step WOM is*

$$C^1_{\text{sum}}(t) = \log_2\left(\left\lceil \frac{t}{2} \right\rceil + 1\right).$$

**Proof.** See Appendix. ■

In the following, we give a code construction[1] for even $t$, which is sum-rate optimal. It also serves as part of the proof for Proposition 4.2.2. Partition the set of all cells into two blocks and write odd messages to one block on odd writes and even messages to the other block on even writes, as shown in Table 4.1. Thus, each block of cells can reliably store $t/2$ messages using a traditional $(t/2)$-write WOM code and decoder $i$ can recover both messages $(M_{i-1}, M_i)$ stored in the two blocks.

**Construction 4.2.1** Let $t$ and $n$ be positive integers, with $t$ even, and let $\lambda \in [0,1]$ such that $\lambda n$ is an integer. Let $\breve{} = 1-\lambda$ and $s = t/2$. Suppose that the cell levels after the $i$-th write, $i \in [1:t]$, are $(\boldsymbol{y}'_i, \boldsymbol{y}''_i)$, where $\boldsymbol{y}'_i$ and $\boldsymbol{y}''_i$ denote blocks of lengths $\lambda n$ and $(1-\lambda)n$, respectively. Let $\mathcal{C}_1$ be a $[\lambda n, s; 2^{\lambda n R'_1}, \ldots, 2^{\lambda n R'_s}]$ WOM code of length $\lambda n$ with encoder $\mathcal{E}'_i(m_i, \boldsymbol{y}'_{i-1})$, $m_i \in [1 : 2^{\lambda n R'_i}]$, on the $i$-th write, $i \in [1:s]$. Let $\mathcal{C}_2$ be a $[\breve{}n, s; 2^{\breve{}nR''_1}, \ldots, 2^{\breve{}nR''_s}]$ WOM code of length $\breve{}n$ with encoder $\mathcal{E}''_i(m_i, \boldsymbol{y}''_{i-1})$, $m_i \in [1 : 2^{\breve{}nR''_i}]$, on the $i$-th write, $i \in [1:s]$. Let $R_{2i-1} = \lambda R'_i$ and $R_{2i} = \breve{}R''_i$, $\forall i \in [1:s]$. An $[n, t; 2^{nR_1}, \ldots, 2^{nR_t}]$ consecutive two-step WOM code $\mathcal{C}$ of length $n$ is constructed as follows. The cells are partitioned into block 1 with length $\lambda n$ and block 2 with length $\breve{}n$. On the $i$-th write, the encoder $i$ assigns the codeword $\boldsymbol{x}_i = (\boldsymbol{x}'_i, \boldsymbol{x}''_i)$ as follows:

1. For odd $i = 2j-1$, write message $m_i \in [1 : 2^{nR_i}]$ to block 1 using the encoder on the $j$-th write from $\mathcal{C}_1$ and keep block 2 unchanged, i.e.,

$$\boldsymbol{x}'_i = \mathcal{E}'_j(m_i, \boldsymbol{y}'_{i-1}).$$

2. For even $i = 2j$, write message $m_i \in [1 : 2^{nR_i}]$ to block 2 using the encoder on the $j$-th write from $\mathcal{C}_2$ and keep block 1 unchanged, i.e.,

$$\boldsymbol{x}''_i = \mathcal{E}''_j(m_i, \boldsymbol{y}''_{i-1}).$$

■

It can be seen that $\mathcal{C}$ is a consecutive two-step WOM code. If $R'_i = R''_i, \forall i \in [1:s]$, the sum-rate of $\mathcal{C}$ in Construction 4.2.1 is $\sum_{i=1}^{t} R_t = \sum_{i=1}^{s} \lambda R'_i + \sum_{i=1}^{s} \breve{}R''_i = \sum_{i=1}^{s} R'_i$. Therefore,

[1] In all the following constructions, the decoders of the WOM codes for Problems 1, 2, and 3 are similar to the decoders of the traditional WOM codes that are assumed to exist in each construction, and thus we omit the details of the decoders here.

if $\mathcal{C}_1$ and $\mathcal{C}_2$ are sum-rate optimal, then $\mathcal{C}$ achieves the sum-capacity $\log_2(\frac{t}{2}+1)$. For odd $t$, a consecutive two-step WOM code can be constructed similarly.

It is readily to generalize Construction 4.2.1 to the problem of consecutive $k$-step WOM, where on the $i$-th write, $i \in [1:t]$, encoder $i$ stores $(M_{i-k+1}, M_{i-k+2}, \ldots, M_i)$ ($M_i = \emptyset, \forall i \leqslant 0$) and decoder $i$ has to recover all of the $k$ messages. Furthermore, the sum-capacity of consecutive $k$-step WOM can be expressed as the following.

**Corollary 4.2.4.** *The sum-capacity $C^1_{\text{sum}}(k,t)$ of the $t$-write consecutive $k$-step WOM is*

$$C^1_{\text{sum}}(k,t) = \log_2\left(\left\lceil \frac{t}{k} \right\rceil + 1\right).$$

## 4.3 Arbitrary Two-Step WOM

In this section we study the arbitrary two-step WOM defined in Problem 2. Note that if a WOM code $\mathcal{C}$ is a $t$-write arbitrary two-step WOM code, then we can construct from it a $t$-write consecutive two-step WOM code. Therefore, the sum-capacity $C^2_{\text{sum}}(t)$ of the arbitrary two-step WOM is upper bounded as $C^2_{\text{sum}}(t) \leqslant C^1_{\text{sum}}(t) = \log_2(\lceil \frac{t}{2} \rceil + 1)$.

Now we give a construction that strictly outperforms the construction in the introduction and achieves $2/3$ of $C^1_{\text{sum}}(t)$ of the consecutive two-step WOM, while keeping track of arbitrary messages as required. Partition the set of all cells into three blocks as illustrated in Table 4.2. In the first two blocks, we write in the exactly same manner as for the consecutive two-step WOM. The third block is updated with $M_{s(i)}$ every other write to help retrieve the desired message of the arbitrary demand. This can be improved by further enlarging the number of blocks as given in Construction 4.3.1.

**Table 4.2**: Writing arrangement of the arbitrary 2-step WOM code

| | block 1 | block 2 | block 3 |
|---|---|---|---|
| 1st write | $M_1$ | | |
| 2nd write | $M_1$ | $M_2$ | $M_{s(2)}$ |
| 3rd write | $M_3$ | $M_2$ | $M_{s(2)}$ |
| 4th write | $M_3$ | $M_4$ | $M_{s(4)}$ |
| 5th write | $M_5$ | $M_4$ | $M_{s(4)}$ |

**Construction 4.3.1** Let $\ell$ and $t$ be positive integers such that $t$ is a multiple of $\ell$. The cells consist of $\ell+1$ blocks, each of size $n'$; thus $n = n'(l+1)$. After the $i$-th write, $i \in [1:t]$, the cell levels are $(\boldsymbol{y}_i^{(1)}, \boldsymbol{y}_i^{(2)}, \ldots, \boldsymbol{y}_i^{(\ell+1)})$, where $\boldsymbol{y}_i^{(j)}, j \in [1:\ell+1]$, denotes the $j$-th block of

length-$n'$ cells. Let $\mathcal{C}_W$ be an $[n', t/\ell; 2^{n'R'_1}, \ldots, 2^{n'R'_{t/\ell}}]$ WOM code of length $n'$ with encoder $\mathcal{E}'_i(m_i, \boldsymbol{y}_{i-1})$, $m_i \in [1 : 2^{n'R'_i}]$, on the $i$-th write, $i \in [1 : t/\ell]$. An $[n, t; 2^{nR_1}, 2^{nR_2}, \ldots, 2^{nR_t}]$ arbitrary two-step WOM code $\mathcal{C}$ is constructed, where $R_i = R'_{\lceil i/\ell \rceil}, i \in [1 : t]$. On the $i$-th write, $i \in [1 : t]$, the encoder $i$ assigns the codeword $\boldsymbol{x}_i = (\boldsymbol{x}_i^{(1)}, \boldsymbol{x}_i^{(2)}, \ldots, \boldsymbol{x}_i^{(\ell+1)})$ using the following rules. Let $h = (i - 1 \bmod \ell)$ and $j = \lceil \frac{i}{\ell} \rceil$.

1. Write message $m_i \in [1 : 2^{n'R_i}]$ to the $(h+1)$-st block, using the encoder on the $j$-th write from $\mathcal{C}_W$ and keep the rest of the first $n'\ell$ cells unchanged, i.e.,

$$\boldsymbol{x}_i^{(h+1)} = \mathcal{E}'_j(m_i, \boldsymbol{y}_{i-1}^{(h+1)}).$$

2. If $h = 0$ and $i \neq 1$, write message $m_{s(i)}$ to the $(\ell+1)$-st block, using the encoder on the $(j-1)$-st write from $\mathcal{C}_W$, i.e.,

$$\boldsymbol{x}_i^{(\ell+1)} = \mathcal{E}'_{j-1}(m_{s(i)}, \boldsymbol{y}_{i-1}^{(\ell+1)}).$$

Otherwise, the last block is kept unchanged.

■

**Proposition 4.3.1.** *If the WOM code $\mathcal{C}_W$ is sum-rate optimal, then the code $\mathcal{C}$ in Construction* 4.3.1 *is an arbitrary two-step WOM code with sum-rate*

$$R^2_{\text{sum}}(t) = \frac{\ell}{\ell+1} \log_2 \left( \frac{t}{\ell} + 1 \right).$$

For large $t$, let $\ell = \log_2 t$. Then the asymptotic sum-rate is

$$R^2_{\text{sum}}(t) = \tfrac{\log_2 t}{\log_2 t + 1} \log_2 (\tfrac{t}{\log_2 t} + 1) = \log_2 t - O(\log_2 (\log_2 t)).$$

Since an upper bound on the sum-capacity is $\log_2 \left( \lceil \frac{t}{2} \rceil + 1 \right)$, this construction is asymptotically optimal in $t$.

If $t$ is not a multiple of $\ell$ we slightly modify Construction 4.3.1. We use a $\lceil \frac{t}{\ell} \rceil$-write WOM code for the first $(t \bmod \ell)$ blocks and a $\lfloor \frac{t}{\ell} \rfloor$-write WOM code for the last $(\ell + 1 - (t \bmod \ell))$ blocks. The constructions yield the following corollary.

**Corollary 4.3.2** *A lower bound of $C^2_{\text{sum}}(t)$ is given by*

$$\max_{\ell \in [1:t]} \frac{(t \bmod \ell) \log_2 (\lceil \frac{t}{\ell} \rceil + 1) + (\ell - (t \bmod \ell)) \log_2 (\lfloor \frac{t}{\ell} \rfloor + 1)}{\ell + 1}.$$

Figure 4.1 shows the upper and lower bounds on the value of $C^2_{\text{sum}}(t)$.

Finally, for the particular case of $t = 3$ we have a different construction achieving sum-rate 1.2 while we can show that an upper bound is 1.5. The construction is presented next.

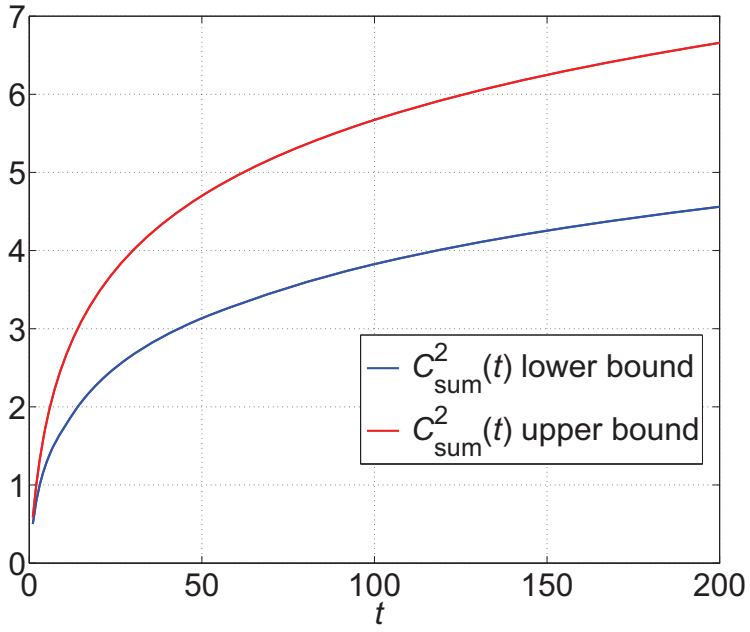


**Figure 4.1**: Bounds on $C^2_{\text{sum}}(t)$

## 4.4 Incremental WOM

We study the incremental WOM model in Problem 3.

**Theorem 4.4.1.** *The sum-capacity of the $t$-write incremental WOM is*

$$C^3_{\text{sum}}(t) = \log_2(t+1).$$

*The optimal trade-off between the common-message sum-rate $R^c_{\text{sum}} := \sum_{i=1}^n R^c_i$ and the private-message sum-rate $R^p_{\text{sum}} := \sum_{i=1}^n R^p_i$ is the set of rate pairs $(R^c_{\text{sum}}, R^p_{\text{sum}})$ such that*

$$R^c_{\text{sum}} \leqslant \prod_{i=1}^{t-1} p_i,$$
$$R^c_{\text{sum}} + R^p_{\text{sum}} \leqslant \prod_{i=1}^{t-1} p_i + \sum_{i=1}^{t-1} \left( \prod_{k=1}^{i-1} p_k \right) H(p_i)$$

*for some $p_1, p_2, \ldots, p_{t-1} \in [\frac{1}{2}, 1]$.*

**Proof.** Theorem 4.4.1 follows by noting that $R^c_1 = R^c_2 = \cdots = R^c_{t-1} = 0$ is optimal for the sum-rate trade-off. ■

Now we focus on the symmetric sum-capacity $C^3_{\text{ssum}}(t)$, defined as the maximum achievable sum-rate when $R^c_1 = R^p_1 = R^c_2 = R^p_2 = \cdots = R^c_t = R^p_t = R$. We denoted by $[n, t; 2^{nR}]$ the $t$-write symmetric incremental WOM code. The following lemma gives an upper bound on the symmetric sum-capacity.

**Lemma 4.4.2.** *The symmetric sum-capacity of the $t$-write incremental WOM is upper bounded as*

$$C^3_{\text{ssum}}(t) \leqslant 2 - \frac{2}{t+1} < 2.$$

**Proof.** Let $\mathcal{C}$ be an $[n, t; (2^{nR}, 2^{nR})]$ symmetric $t$-write incremental WOM code. On each write, the rate of all recoverable information is not greater than 1. In particular, on the $t$-th write, the $t+1$ messages $M^c_1, M^c_2, \ldots, M^c_t, M^p_t$ are recoverable. Therefore, $(t+1)R \leqslant 1$, which gives $R \leqslant \frac{1}{t+1}$, and hence $2tR \leqslant \frac{2t}{t+1} = 2 - \frac{2}{t+1} < 2$. ■

In the following, we give a construction of $t$-write symmetric incremental WOM codes. To illustrate the basic idea, we show a construction for $t = 3$. Suppose that every private/common message represents $k = nR$ bits. Partition the $n$ cells into three blocks. Write $M^c_1$ to the first block. Partition $M^p_1$ into two messages $M^p_{11}$ with $\lambda k$ bits and $M^p_{12}$ with $(1-\lambda)k$ bits. Write $M^p_{11}$ and $M^c_2$ to the second block and $M^p_{12}, M^p_2, (M^c_3, M^p_3)$ to the third block. Thus, in the first block we use a one-write WOM code, in the second block we use a two-write WOM code and in the third block we use a three-write WOM code, as illustrated in Table 4.3.

**Table 4.3**: Writing arrangement of the 3-write incremental WOM code

| | block 1 | block 2 | block 3 |
|---|---|---|---|
| 1st write | $M^c_1$ | $M^p_{11}$ | $M^p_{12}$ |
| 2nd write | $M^c_1$ | $M^c_2$ | $M^p_2$ |
| 3rd write | $M^c_1$ | $M^c_2$ | $(M^c_3, M^p_3)$ |

Suppose that the lengths of the first, second, and third blocks are $n_1, n_2$, and $n_3$, respectively. For the fixed $k$, the problem of maximizing the symmetric sum-rate is identical to minimizing the value of $n = n_1 + n_2 + n_3$ as a function of $\lambda$. Now we state the construction formally and then present the symmetric sum-rate analysis.

**Construction 4.4.1** Let $k$ be a positive integer, $\lambda \in [0, 1]$, and $n_1 = k$. Suppose that the cell levels after the $i$-th write are $(\boldsymbol{y}'_i, \boldsymbol{y}''_i, \boldsymbol{y}'''_i)$, where $\boldsymbol{y}'_i, \boldsymbol{y}''_i$, and $\boldsymbol{y}'''_i$ denote blocks of length $n_1, n_2$, and $n_3$, respectively. Let $\mathcal{C}_1$ be an $[n_1, 1; 2^k]$ WOM code with encoder $\mathcal{E}'_1(m_1)$ for the first write,

$\mathcal{C}_2$ be an $[n_2, 2; 2^{\lambda k}, 2^k]$ WOM code with encoders $\mathcal{E}_i''(m_i, \boldsymbol{y}_{i-1}'')$, $i \in [1:2]$, for the first two writes, and $\mathcal{C}_3$ be an $[n_3, 3; 2^{(1-\lambda)k}, 2^k, 2^{2k}]$ WOM code with encoder $\mathcal{E}_i'''(m_i, \boldsymbol{y}_{i-1}''')$, $i \in [1:3]$, for all three writes. An $[n, 3; (2^k, 2^k)]$ symmetric incremental WOM code $\mathcal{C}$ is constructed. On the $i$-th write, encoder $i$ assigns the codeword $\boldsymbol{x}_i = (\boldsymbol{x}_i', \boldsymbol{x}_i'', \boldsymbol{x}_i''')$ using the following encoding rules:

1. If $i = 1$, then write message $m_1^c \in [1 : 2^k]$ to block 1 using the encoder from $\mathcal{C}_1$, write message $m_{11}^p \in [1 : 2^{\lambda k}]$ to block 2 using the encoder for the first write from $\mathcal{C}_2$, and write message $m_{12}^p \in [1 : 2^{(1-\lambda)k}]$ to block 3 using the encoder for the first write from $\mathcal{C}_3$, i.e.,

$$(\boldsymbol{x}_1', \boldsymbol{x}_1'', \boldsymbol{x}_1''') = \left(\mathcal{E}_1'(m_1^c), \mathcal{E}_1''\left(m_{11}^p\right), \mathcal{E}_1'''\left(m_{12}^p\right)\right).$$

2. If $i = 2$, then the first $n_1$ cells are unchanged, write message $m_2^c \in [1 : 2^k]$ to block 2 using the encoder for the second write from $\mathcal{C}_2$, and write message $m_2^p \in [1 : 2^k]$ to block 3 using the encoder for the second write from $\mathcal{C}_3$, i.e.,

$$(\boldsymbol{x}_2'', \boldsymbol{x}_2''') = \left(\mathcal{E}_2''\left(m_2^c, \boldsymbol{y}_1''\right), \mathcal{E}_2'''\left(m_2^p, \boldsymbol{y}_1'''\right)\right).$$

3. If $i = 3$, then the first $n_1 + n_2$ cells are unchanged and write message $(m_3^c, m_3^p) \in [1 : 2^{2k}]$ to block 3, using the encoder for the third write from $\mathcal{C}_3$, i.e.,

$$\boldsymbol{x}_3''' = \mathcal{E}_3'''\left((m_3^c, m_3^p), \boldsymbol{y}_2'''\right).$$

■

The symmetric sum-rate of the code $\mathcal{C}$ is given by $6k/n$. As $k$ is fixed, this value is maximized when $n$ is minimized. We denote by $n_2(\lambda)$ the minimum length of an $[n_2, 2; 2^{\lambda k}, 2^k]$ WOM code and similarly $n_3(\lambda)$ is the minimum length of an $[n_3, 3; 2^{(1-\lambda)k}, 2^k, 2^{2k}]$ WOM code. Then, the problem is to find the value of $\min_{\lambda \in [0,1]}(n_2(\lambda) + n_3(\lambda))$.

**Proposition 4.4.3.** *The minimum value of $n$ in Construction 4.4.1 is $n = 4.386k$, which is achieved by setting $\lambda = 0.3116$. The corresponding symmetric sum-rate is $R_{\text{ssum}}^3(3) = 1.3679$.*

**Proof.** See Appendix. ■

We are now ready to generalize the construction for an arbitrary number of writes $t$. Each of the messages $M_1^c, M_1^p, \ldots, M_t^c, M_t^p$ represents $k = nR$ bits, and the $n$ cells are partitioned

**Table 4.4**: Writing arrangement of the $t$-write incremental WOM code

| | block 1 | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | $\cdots$ | block $t$ |
|---|---|---|---|---|---|---|---|
| 1 | $M_1^c$ | $M_{11}^p$ | $M_{12}^p$ | $\cdots$ | $\cdots$ | $M_{1,t-2}^p$ | $M_{1,t-1}^p$ |
| 2 | $M_1^c$ | $M_2^c$ | $M_{21}^p$ | $\cdots$ | $\cdots$ | $M_{2,t-3}^p$ | $M_{2,t-2}^p$ |
| 3 | $M_1^c$ | $M_2^c$ | $M_3^c$ | $M_{31}^p$ | $\cdots$ | $M_{3,t-4}^p$ | $M_{3,t-3}^p$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\ddots$ | $\vdots$ | $\vdots$ |
| | $M_1^c$ | $M_2^c$ | $M_3^c$ | $M_4^c$ | $\cdots$ | $M_{t-2,1}^p$ | $M_{t-2,2}^p$ |
| | $M_1^c$ | $M_2^c$ | $M_3^c$ | $M_4^c$ | $\cdots$ | $M_{t-1}^c$ | $M_{t-1}^p$ |
| $t$ | $M_1^c$ | $M_2^c$ | $M_3^c$ | $M_4^c$ | $\cdots$ | $M_{t-1}^c$ | $(M_t^c, M_t^p)$ |

into $t$ blocks. Message $M_i^p, i \in [1 : t-2]$, is partitioned into $t-i$ parts $(M_{i1}^p, M_{i2}^p, \ldots, M_{i,t-i}^p)$. The arrangement of these messages when written into the memory is depicted in Table 4.4.

According to this layout, the $i$-th block, for $i \in [1 : t]$, consists of $n_i$ cells and is used to construct an $i$-write WOM code. Assume that message $M_{ij}^p$ for $i \in [1 : t-2], j \in [1 : t-i]$ represents $\lambda_{i,j}k$ bits, where $\sum_{j=1}^{t-i} \lambda_{i,j} = 1$. Then for $i \in [2 : t-1]$, messages $(M_{1,i-1}^p, M_{2,i-2}^p, \ldots, M_{i-1,1}^p, M_i^c)$ will be written as an $i$-write WOM code of length $n_i$, and messages $\left(M_{1,t-1}^p, M_{2,t-2}^p, \ldots, M_{t-2,2}^p, M_{t-1}^p, (M_t^c, M_t^p)\right)$ will be written as a $t$-write WOM code of length $n_t$, where $(M_t^c, M_t^p)$ represents $2k$ bits.

Similarly to the three-write case, the symmetric sum-rate of the $t$-write incremental WOM code is $R_{\text{ssum}}^3(t) = \frac{2tk}{n}$. If $k$ is fixed, $R_{\text{ssum}}^3(t)$ is maximized when $n$ is minimized. For $j \in [2 : t]$, let $\boldsymbol{\lambda}_j$ be a length-$(j-1)$ vector defined as

$$\boldsymbol{\lambda}_j = (\lambda_{1,j-1}, \lambda_{2,j-2}, \ldots, \lambda_{j-1,1}) \in [0,1]^{j-1},$$

where $\sum_{h=1}^{t-i} \lambda_{i,h} = 1, \forall i \in [1 : t-1]$. Let $n_1 = k$ be fixed and we denote by $n_j(\boldsymbol{\lambda}_j)$ to be the minimum length of an $[n_j, j; 2^{\lambda_{1,j-1}k}, 2^{\lambda_{2,j-2}k}, \ldots, 2^{\lambda_{j-1,1}k}, 2^k]$ WOM code for $j \in [2 : t-1]$ and $n_t(\boldsymbol{\lambda}_t)$ is the minimum length of an $[n_t, t; 2^{\lambda_{1,t-1}k}, 2^{\lambda_{2,t-2}k}, \ldots, 2^{\lambda_{t-2,2}k}, 2^k, 2^{2k}]$ WOM code. According to Table 4.4, we seek to find the minimum total block length $N_t^*$, which is equivalent to solving the following minimization problem

$$\text{Find } N_t^* = \min_{\boldsymbol{\lambda}_2, \ldots, \boldsymbol{\lambda}_t} \left\{ n_1 + \sum_{j=2}^{t} n_j(\boldsymbol{\lambda}_j) \right\}, \tag{P1}$$

subject to $\lambda_{i,h} \in [0,1], \sum_{h=1}^{t-i} \lambda_{i,h} = 1, \forall i \in [1 : t-1], h \in [1 : t-i]$.

The solution to Problem (P1) is closely related to the following minimization problem.

**Definition 4.4.4.** *Let* $\boldsymbol{p}_i = (p_{i,i+1}, p_{i,i+2}, \ldots, p_{i,t}) \in [0, \frac{1}{2}]^{t-i}$, *for* $i \in [1 : t-1]$, *be* $t-1$

*vectors of decreasing length. Let*

$$f_t(\boldsymbol{p}_1,\ldots,\boldsymbol{p}_{t-1}) \stackrel{def}{=} 1+\left(\sum_{j=2}^{t-1}\frac{1}{\prod_{\ell=1}^{j-1}(1-p_{\ell,j})}\right)+\frac{2}{\prod_{\ell=1}^{t-1}(1-p_{\ell,t})},$$

*and define the minimization problem*

$$\text{minimize} \quad f_t(\boldsymbol{p}_1,\ldots,\boldsymbol{p}_{t-1}), \tag{P2}$$

*with respect to* $\boldsymbol{p}_1,\ldots,\boldsymbol{p}_{t-1}$, *subject to the following conditions*

$$p_{i,h}\in[0,\frac{1}{2}],\forall i\in[1:t-1],h\in[i+1:t],$$

*and*

$$\left(\sum_{h=i+1}^{t-1}\frac{h(p_{i,h})}{\prod_{s=i}^{h-1}(1-p_{s,h})}\right)+\frac{h(p_{i,t})}{\left(\prod_{s=i}^{t-2}(1-p_{s,t})\right)h(p_{t-1,t})}=1,$$

*for all* $i\in[1:t-2]$, *where* $p_{t-1,t}$ *satisfies* $\frac{h(p_{t-1,t})}{1-p_{t-1,t}}=\frac{1}{2}$.

*Let* $F_t^*$ *be the minimum value of Problem* (P2). *A vector* $(\boldsymbol{p}_1,\ldots,\boldsymbol{p}_{t-1})$ *is called an* **optimal solution** *or simply* **optimal** *to problem* (P2) *if* $f_t(\boldsymbol{p}_1,\ldots,\boldsymbol{p}_{t-1})=F_t^*$.

**Lemma 4.4.5.** *The minimum block length* $N_t^*$ *for Problem* (P1) *is*

$$N_t^*=kF_t^*,$$

*Furthermore, the symmetric sum-capacity of incremental WOM satisfies* $C_{\text{ssum}}^3(t)\geqslant\frac{2t}{F_t^*}$.

**Proof.** See Appendix. ■

According to Lemma 4.4.5, we are left to solve Problem (P2). Theorem 4.4.6 recursively solves Problem (P2). Let $\hat{\boldsymbol{p}}_i=(\hat{p}_{i,i+1},\hat{p}_{i,i+1},\ldots,\hat{p}_{i,t+1})$, for $i\in[1:t]$, be optimal for the $(t+1)$-write incremental WOM code, and let $\tilde{\boldsymbol{p}}_i=(\tilde{p}_{i,i+1},\ldots,\tilde{p}_{i,t})$ for $i\in[1:t-1]$, be optimal for the $t$-write incremental WOM code. Then an optimal $(\hat{\boldsymbol{p}}_1,\ldots,\hat{\boldsymbol{p}}_t)$ can be derived from the optimal $(\tilde{\boldsymbol{p}}_1,\ldots,\tilde{\boldsymbol{p}}_{t-1})$. Since the solution when $t=3$ is known, the solution to Problem (P2) for arbitrary $t$ can be derived accordingly.

**Theorem 4.4.6.** *Let* $\hat{\boldsymbol{p}}_i=(\hat{p}_{i,i+1},\hat{p}_{i,i+2},\ldots,\hat{p}_{i,t+1})\in[0,\frac{1}{2}]^{t-i+1}$ *for* $i\in[1:t]$. *Let* $\tilde{\boldsymbol{p}}_i=(\tilde{p}_{i,i+1},\ldots,\tilde{p}_{i,t})\in[0,\frac{1}{2}]^{t-i}$ *for* $i\in[1:t-1]$. *Suppose* $(\tilde{\boldsymbol{p}}_1,\ldots,\tilde{\boldsymbol{p}}_{t-1})$ *is optimal, then there exists an optimal* $(\hat{\boldsymbol{p}}_1,\ldots,\hat{\boldsymbol{p}}_t)$, *such that*

$$\hat{\boldsymbol{p}}_i=\tilde{\boldsymbol{p}}_{i-1},\forall i\in[2:t],$$

*and* $\hat{\boldsymbol{p}}_1=(p_{1,2},\ldots,p_{1,t})$, *where* $\forall j\in[2:t]$, $p_{1,j}$ *satisfies* $\frac{h(p_{1,j})}{1-p_{1,j}}=\frac{1}{F_t^*}$.

**Proof.** See Appendix. ■

Figure 4.2 shows the achievable symmetric sum-rate of the time-sharing scheme described in the introduction and our construction based on the optimal partition strategy, $\lambda_{i,j}$, $i \in [1 : t-2]$, $j \in [1 : t-i]$, that maximizes $R^3_{\text{ssum}}(t)$.

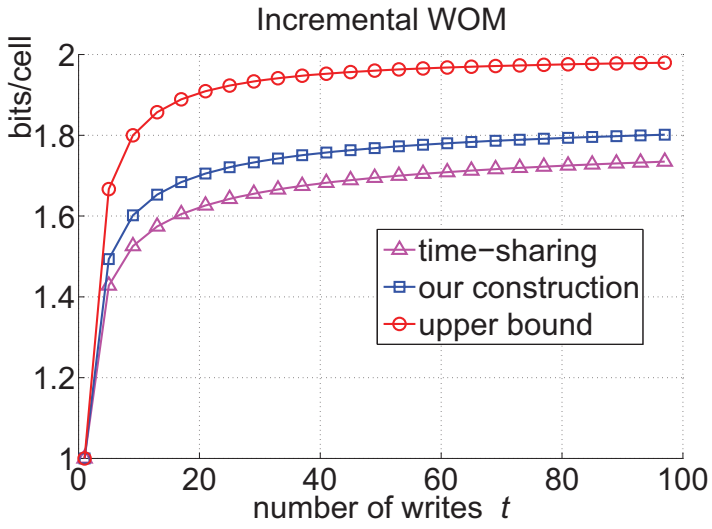


**Figure 4.2**: Lower and upper bounds on $C^3_{\text{ssum}}(t)$

## 4.5 Conclusion

Write-once memory (WOM) is a binary storage medium in which each memory cell is initially in state 0 and can be irreversibly programmed to state 1. This chapter studies the problem of writing multiple messages into a WOM. Instead of writing a new message (and obliterating old ones) as in the traditional setup, the user wishes to retain access to some of the previously written messages. The capacity region is studied and code constructions are proposed for three canonical cases.

## 4.6 Appendix A

**Proof of Proposition 4.4.3.** Let us first find the value of $n_2(\lambda)$. That is, we find a WOM code of minimum length $n_2(\lambda)$ such that its rate on the first write is $R_1 = \lambda k/n_2(\lambda)$ and its rate on the second write is $R_2 = k/n_2(\lambda)$. Thus, we have $R_1/R_2 = \lambda$. Since the capacity region

of the two-write WOM is given by $\{(R_1, R_2)|R_1 \leqslant h(p_1), R_2 \leqslant p_1, \text{ for some } p_1 \in [1/2, 1]\}$, and we find a WOM code of minimum length, the ratio of $R_1$ and $R_2$ satisfies

$$h(p_1)/p_1 = \lambda, \tag{4.1}$$

for some $p_1 \in [1/2, 1]$. Note that if $\lambda$ is positive then equation (4.1) always has a solution, which we denote by $p_1(\lambda)$. Now, we deduce from $R_2 = k/n_2 = p_1$ that $n_2(\lambda) = k/p_1(\lambda)$.

Similarly, the capacity region of the three-write WOM is given by $\{(R_1, R_2, R_3)|R_1 \leqslant h(p_2), R_2 \leqslant p_2 h(p_3), R_3 \leqslant p_2 p_3, \text{ for some } p_2, p_3 \in [1/2, 1]\}$. Thus, it can be shown that the values of $p_2, p_3 \in [1/2, 1]$ that give the minimum code length for $n_3(\lambda)$ satisfy

$$h(p_2)/(p_2 h(p_3)) = 1 - \lambda, \tag{4.2}$$

$$h(p_3)/p_3 = 1/2. \tag{4.3}$$

The value of $p_3$ is independent of $\lambda$ and is given by $p_3 = 0.9055$, and $p_2(\lambda)$ is the solution to equation (4.2). Hence, $n_3(\lambda)$ satisfies $n_3(\lambda) = 2k/((p_2(\lambda)p_3)$.

We are now left to solve the minimization problem

$$\text{minimize } \left(\frac{1}{p_1(\lambda)} + \frac{2}{p_2(\lambda)p_3}\right), \tag{4.4}$$

with $\lambda \in [0, 1]$, where $p_1(\lambda), p_2(\lambda)$, and $p_3$ satisfy equations (4.1), (4.2), and (4.3) respectively.

From Equation (4.3), $p_3$ was already calculated numerically. From Equation (4.1) and (4.2), we have

$$\frac{h(p_1(\lambda))}{p_1(\lambda)} + \frac{h(p_2(\lambda))}{p_2(\lambda)h(p_3)} = 1.$$

Therefore, we can formulate the minimization problem as

$$\text{minimize } \left(\frac{1}{p_1} + \frac{2}{p_2 p_3}\right)$$

with $p_1, p_2 \in [\frac{1}{2}, 1]$, subject to

$$\frac{h(p_1)}{p_1} + \frac{h(p_2)}{p_2 h(p_3)} = 1, \text{ where } p_3 = 0.9055.$$

It follows that $p_1 = p_2 = 0.9479$ and we get $\lambda = h(p_1)/p_1 = h(p_3)/(h(p_3) + 1) = 0.3116$. Therefore, $n = n_1 + n_2(\lambda) + n_3(\lambda) = 4.386k$ and $R^3_{\text{ssum}}(3)$ satisfies

$$R^3_{\text{ssum}}(3) = 6R = \frac{6k}{n} = \frac{6k}{n_1 + n_2(\lambda) + n_3(\lambda)} = 1.3679.$$

This completes the proof. ■

**Proof of Lemma 4.4.5.** First, remember that the capacity region of a $j$-write WOM is

$$\mathscr{C}_{\text{WOM}}(j) = \Big\{(R_1, \ldots, R_j) | R_1 \leqslant h(p_{1,j}), R_2 \leqslant (1-p_{1,j})h(p_{2,j}), \\ \ldots, R_{j-1} \leqslant \Big(\prod_{\ell=1}^{j-2}(1-p_{\ell,j})\Big)h(p_{j-1,j}), R_j \leqslant \prod_{\ell=1}^{j-1}(1-p_{\ell,j}), \\ \text{where } 0 \leqslant p_{1,j}, \ldots, p_{j-1,j} \leqslant 1/2\Big\}.$$

In this particular construction, the $j$-th block for $j \in [2 : t-1]$ of length $n_j$ is an $[n_j, j; 2^{\lambda_{1,j-1}k}, 2^{\lambda_{2,j-2}k}, \ldots, 2^{\lambda_{j-1,1}k}, 2^k]$ WOM code. Thus, the ratio between the rate of the $\ell$-th write, $R_\ell$, and the last write, $R_j$, is $\frac{R_\ell}{R_j} = \lambda_{\ell,j-\ell}, \forall \ell \in [1 : j-1]$. If such a WOM code has minimum length $n_j(\boldsymbol{\lambda}_j)$, then all inequalities on $R_\ell, \forall \ell \in [1 : j]$ in the capacity region $\mathscr{C}_{\text{WOM}}(j)$ turn to equality, and therefore $\frac{R_\ell}{R_j} = \frac{h(p_{\ell,j})}{\prod_{s=\ell}^{j-1}(1-p_{s,j})}$. Thus we have

$$\lambda_{\ell,j-\ell} = \frac{h(p_{\ell,j})}{\prod_{s=\ell}^{j-1}(1-p_{s,j})}, \forall j \in [2 : t-1], \ell \in [1 : j-1],$$

and by substitution of indices we have

$$\lambda_{\ell,j} = \frac{h(p_{\ell,\ell+j})}{\prod_{s=\ell}^{\ell+j-1}(1-p_{s,\ell+j})}, \forall j \in [1 : t-2], \ell \in [1 : t-j].$$

From the definition of the vectors $\boldsymbol{\lambda}_j$ for $j \in [2 : t]$, the values of $p_{1,j}, \ldots, p_{j-1,j}$ are a function of only the entries in the vector $\boldsymbol{\lambda}_j$, and thus we can denote $p_{\ell,j}(\boldsymbol{\lambda}_j), \forall j \in [2 : t-1], \ell \in [1 : j-1]$.

Moreover, since on the last write $k$ bits are written, we get $R_j = \frac{k}{n_j(\boldsymbol{\lambda}_j)} = \prod_{\ell=1}^{j-1}(1 - p_{\ell,j}(\boldsymbol{\lambda}_j))$. Thus,

$$n_j(\boldsymbol{\lambda}_j) = \frac{k}{\prod_{\ell=1}^{j-1}(1-p_{\ell,j}(\boldsymbol{\lambda}_j))}.$$

Similarly, for $j = t$, the $t$-th block is an $[n_t, t; 2^{\lambda_{1,t-1}k}, 2^{\lambda_{2,t-2}k}, \ldots, 2^{\lambda_{t-2,2}k}, 2^k, 2^{2k}]$ WOM code. The ratio between the rate of the $\ell$-th write and the $(t-1)$-st write is $\frac{R_\ell}{R_{t-1}} = \lambda_{\ell,t-\ell}, \forall \ell \in [1 : t-2]$. For such a WOM code with minimum length $n_t(\boldsymbol{\lambda}_t)$, we have

$$\lambda_{\ell,t-\ell} = \frac{R_\ell}{R_{t-1}} = \frac{h(p_{\ell,t})}{\prod_{s=\ell}^{t-2}(1-p_{s,t})h(p_{t-1,t})}, \forall \ell \in [1 : t-2]$$

and thus $p_{\ell,t}$, which is a function of $\boldsymbol{\lambda}_t$, is denoted by $p_{\ell,t}(\boldsymbol{\lambda}_t)$. The value of $p_{t-1,t}$ satisfies $\frac{h(p_{t-1,t})}{1-p_{t-1,t}} = \frac{1}{2}$, and for the simplicity of the coming notation will be denoted by $p_{t-1,t}(\boldsymbol{\lambda}_t)$. Therefore, as before, we get that

$$n_t(\boldsymbol{\lambda}_t) = \frac{2k}{\prod_{\ell=1}^{t-1}(1-p_{\ell,t}(\boldsymbol{\lambda}_t))}.$$

Now we get that that total number of cell is expressed by

$$\begin{aligned} n &= k + \sum_{j=2}^{t} n_j(\boldsymbol{\lambda}_j) \\ &= k + \left( \sum_{j=2}^{t-1} \frac{k}{\prod_{\ell=1}^{j-1}(1-p_{\ell,j}(\boldsymbol{\lambda}_j))} \right) + \frac{2k}{\prod_{\ell=1}^{t-1}(1-p_{\ell,t}(\boldsymbol{\lambda}_t))}, \end{aligned}$$

where the constraints on $(\boldsymbol{\lambda}_2, \ldots, \boldsymbol{\lambda}_t)$ are

$$\sum_{h=1}^{t-i} \lambda_{i,h} = 1, \forall i \in [1 : t-2],$$

and the constraint $\lambda_{t-1,1} = 1$ has already been satisfied in the code construction.

Therefore, $\forall i \in [1 : t-2]$,

$$\begin{aligned} \sum_{h=1}^{t-i} \lambda_{i,h} &= \sum_{h=1}^{t-i-1} \lambda_{i,h} + \lambda_{i,t-i} \\ &= \sum_{h=1}^{t-i-1} \frac{h(p_{i,i+h})}{\prod_{s=i}^{i+h-1}(1-p_{s,i+h})} + \frac{h(p_{i,t})}{\prod_{s=i}^{t-2}(1-p_{s,t})h(p_{t-1,t})} \\ &= \left( \sum_{h=i+1}^{t-1} \frac{h(p_{i,h})}{\prod_{s=i}^{h-1}(1-p_{s,h})} \right) + \frac{h(p_{i,t})}{\left(\prod_{s=i}^{t-2}(1-p_{s,t})\right) h(p_{t-1,t})} \\ &= 1, \end{aligned}$$

and $p_{t-1,t}$ satisfies $\frac{h(p_{t-1,t})}{1-p_{t-1,t}} = \frac{1}{2}$.

Finally, we conclude that in order to minimize the block length $n$ under the constraints on $(\boldsymbol{\lambda}_2, \ldots, \boldsymbol{\lambda}_t)$, it is suffice the minimize the function $f_t(\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{t-1})$ under the constraints on $(\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{t-1})$. Furthermore, we get that $N_t^* = kF_t^*$, and hence $C_{\text{sym}}^3(t) \geqslant \frac{2t}{F_t^*}$.

■

**Proof of Theorem 4.4.6.** First we prove the following lemma.

**Lemma 4.6.1.** *Suppose $(\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{t-1})$ is optimal for Problem* (P2)*, i.e., $f_t(\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{t-1}) = F_t^*$, then*

$$p_{1,r} = p_1, \forall r \in [2 : t],$$

*where $p_1$ is a constant.*

**Proof.** There are $t-2$ constraints on $(\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{t-1})$, that is, for $p_{i,h} \in [0, \frac{1}{2}], \forall i \in [1 : t-1], h \in [i+1 : t]$, they should satisfy

$$\left( \sum_{h=i+1}^{t-1} \frac{h(p_{i,h})}{\prod_{s=i}^{h-1}(1-p_{s,h})} \right) + \frac{h(p_{i,t})}{\left(\prod_{s=i}^{t-2}(1-p_{s,t})\right) h(p_{t-1,t})} = 1,$$

for all $i \in [1 : t-2]$, where $p_{t-1,t}$ satisfies $\frac{h(p_{t-1,t})}{1-p_{t-1,t}} = \frac{1}{2}$. We denote the $t-2$ constraints by Constraint $\ell$, for $\ell \in [1 : t-2]$.

Using $t-2$ Lagrange multipliers $(\gamma_1, \ldots, \gamma_{t-2})$, let

$$\begin{aligned} L(\gamma_1, \ldots, \gamma_{t-2}) &= f_t(\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{t-1}) \\ &+ \sum_{i=1}^{t-2} \gamma_i \Big( \sum_{h=i+1}^{t-1} \frac{h(p_{i,h})}{\prod_{s=i}^{h-1}(1-p_{s,h})} + \frac{h(p_{i,t})}{\left(\prod_{s=i}^{t-2}(1-p_{s,t})\right) h(p_{t-1,t})} \Big) \end{aligned}$$

Note that only Constraint 1 involves $p_{1,r}$, for $r \in [2 : t]$, and

$$\left( \frac{h(x)}{1-x} \right)' = \frac{-\log(x)}{(1-x)^2}.$$

Taking partial derivative respective to $p_{1,r}$, for $r = [2 : t-1]$, we have

$$\begin{aligned} \frac{\partial L}{\partial p_{1,r}} &= \frac{\partial f_t(\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{t-1})}{\partial p_{1,r}} \\ &+ \gamma_1 \frac{\partial}{\partial p_{1,r}} \Big( \sum_{h=2}^{t-1} \frac{h(p_{1,h})}{\prod_{s=1}^{h-1}(1-p_{s,h})} + \frac{h(p_{1,t})}{\left(\prod_{s=1}^{t-2}(1-p_{s,t})\right) h(p_{t-1,t})} \Big) \\ &= \frac{\partial f_t(\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{t-1})}{\partial p_{1,r}} + \gamma_1 \frac{\partial}{\partial p_{1,r}} \sum_{h=2}^{t-1} \frac{h(p_{1,h})}{\prod_{s=1}^{h-1}(1-p_{s,h})}, \end{aligned}$$

where

$$\begin{aligned} &\frac{\partial f_t(\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{t-1})}{\partial p_{1,r}} \\ =&\frac{\partial}{\partial p_{1,r}} \left( \sum_{j=2}^{t-1} \frac{1}{\prod_{\ell=1}^{j-1}(1-p_{\ell,j})} + \frac{2}{\prod_{\ell=1}^{t-1}(1-p_{\ell,t})} \right) \\ =&\frac{\partial}{\partial p_{1,r}} \sum_{j=2}^{t-1} \frac{1}{\prod_{\ell=1}^{j-1}(1-p_{\ell,j})} \\ =&\frac{\partial}{\partial p_{1,r}} \frac{1}{\prod_{\ell=1}^{r-1}(1-p_{\ell,r})} \\ =&\frac{1}{(1-p_{1,r})^2 \prod_{\ell=2}^{r-1}(1-p_{\ell,r})}, \end{aligned}$$

and

$$\begin{aligned}&\frac{\partial}{\partial p_{1,r}}\sum_{h=2}^{t-1}\frac{h(p_{1,h})}{\prod_{s=1}^{h-1}(1-p_{s,h})}\\=&\frac{\partial}{\partial p_{1,r}}\frac{h(p_{1,r})}{\prod_{s=1}^{r-1}(1-p_{s,r})}\\=&\frac{-\log(p_{1,r})}{(1-p_{1,r})^2\prod_{\ell=2}^{r-1}(1-p_{\ell,r})}.\end{aligned}$$

Therefore, for $r=[2:t-1]$,

$$\frac{\partial L}{\partial p_{1,r}}=\frac{(1-\gamma_1\log_2 p_{1,r})}{(1-p_{1,r})^2\prod_{j=2}^{r-1}(1-p_{j,r})}\overset{\text{SET}}{=}0.$$

Then $p_{1,r}=2^{-\gamma_1}=p_1,\forall r\in[2:t-1]$.

Similarly, for $p_{1,t}$,

$$\begin{aligned}\frac{\partial L}{\partial p_{1,t}}=&\frac{2}{\left((1-p_{1,t})^2\prod_{j=2}^{t-1}(1-p_{j,t})\right)}\\&-\frac{\gamma_1\log_2 p_{1,t}}{h(p_{t-1,t})\left((1-p_{1,t})^2\prod_{j=2}^{t-2}(1-p_{j,t})\right)}\\\overset{\text{SET}}{=}&0,\end{aligned}$$

Note that $\frac{h(p_{t-1,t})}{1-p_{t-1,t}}=\frac{1}{2}$, we have $\gamma_1=\frac{1}{\log_2 p_{1,t}}$. Therefore, $p_{1,r}=2^{-\gamma_1}=p_1,\forall r\in[2:t]$. ■

Let $\hat{\boldsymbol{p}}_i=(\hat{p}_{i,i+1},\hat{p}_{i,i+1},\ldots,\hat{p}_{i,t+1})$, for $i\in[1:t]$. We would like to find

$$F_{t+1}^*=\min_{\hat{\boldsymbol{p}}_1,\cdots,\hat{\boldsymbol{p}}_t}f_{t+1}(\hat{\boldsymbol{p}}_1,\ldots,\hat{\boldsymbol{p}}_t),$$

subject to $p_{i,h}\in[0,\frac{1}{2}],\forall i\in[1:t],h\in[i+1:t+1]$, and

$$\left(\sum_{h=i+1}^{t}\frac{h(p_{i,h})}{\left(\prod_{s=i}^{h-1}(1-p_{s,h})\right)}\right)+\frac{h(p_{i,t+1})}{\left(\prod_{s=i}^{t-1}(1-p_{s,t+1})\right)h(p_{t,t+1})}=1,$$

for all $i\in[1:t-2]$, where $p_{t,t+1}$ satisfies $\frac{h(p_{t,t+1})}{1-p_{t,t+1}}=\frac{1}{2}$.

It has been proved that in order to achieve the minimum, we have $\hat{p}_{1,r}=p_1,\forall r\in[2:$

$t+1]$, where $p_1$ is a constant. Therefore,

$$\begin{aligned}
&\min_{\hat{\boldsymbol{p}}_1,\ldots,\hat{\boldsymbol{p}}_t} f_{t+1}(\hat{\boldsymbol{p}}_1,\ldots,\hat{\boldsymbol{p}}_t)\\
=&\min_{\hat{\boldsymbol{p}}_1,\ldots,\hat{\boldsymbol{p}}_t} 1+\left(\sum_{j=2}^{t}\frac{1}{\prod_{\ell=1}^{j-1}(1-\hat{p}_{\ell,j})}\right)+\frac{2}{\prod_{\ell=1}^{t}(1-\hat{p}_{\ell,t+1})}\\
=&1+\min_{p_1,\hat{\boldsymbol{p}}_2,\ldots,\hat{\boldsymbol{p}}_t}\frac{\left(1+\sum_{j=3}^{t}\frac{1}{\prod_{\ell=2}^{j-1}(1-\hat{p}_{\ell,j})}+\frac{2}{\prod_{\ell=2}^{t}(1-\hat{p}_{\ell,t})}\right)}{1-p_1}\\
=&1+\min_{p_1,\tilde{\boldsymbol{p}}_1,\ldots,\tilde{\boldsymbol{p}}_{t-1}}\frac{1}{1-p_1}f_t(\tilde{\boldsymbol{p}}_1,\ldots,\tilde{\boldsymbol{p}}_{t-1}),
\end{aligned}$$

where $\tilde{\boldsymbol{p}}_i=(\tilde{p}_{i,i+1},\ldots,\tilde{p}_{i,t})$ for $i\in[1:t-1]$, and

$$\tilde{\boldsymbol{p}}_{i-1}=\hat{\boldsymbol{p}}_i,\forall i\in[2:t],$$

According to Constraint 1 and considering that $\frac{h(\hat{p}_{t,t+1})}{1-\hat{p}_{t,t+1}}=\frac{1}{2}$, we have

$$\frac{h(p_1)}{1-p_1}f_t(\tilde{\boldsymbol{p}}_1,\ldots,\tilde{\boldsymbol{p}}_{t-1})=1.$$

Therefore, the problem is formulated as finding

$$F_{t+1}^{*}=1+\min_{p_1\in[0,\frac{1}{2}]}\frac{1}{h(p_1)}, \tag{P3}$$

subject to $\tilde{p}_{i,h}\in[0,\frac{1}{2}],\forall i\in[1:t-1],h\in[i+1:t]$,

$$\frac{h(p_1)}{1-p_1}f_t(\tilde{\boldsymbol{p}}_1,\ldots,\tilde{\boldsymbol{p}}_{t-1})=1, \tag{1}$$

and

$$\left(\sum_{h=i+1}^{t-1}\frac{h(\tilde{p}_{i,h})}{\prod_{s=i}^{h-1}(1-\tilde{p}_{s,h})}\right)+\frac{h(\tilde{p}_{i,t})}{\left(\prod_{s=i}^{t-2}(1-\tilde{p}_{s,t})\right)h(\tilde{p}_{t-1,t})}=1, \tag{2}$$

for $i\in[1:t-2]$.

Note that in Problem (P3), $\frac{1}{h(p_1)}$ is a decreasing function for $p_1\in(0,0.5]$. Then we would like to find the maximum $p_1\in(0,0.5]$ under the constraints of Equation (1) and Equation (2). From Equation (1), we have $\frac{h(p_1)}{1-p_1}=\frac{1}{f_t(\tilde{\boldsymbol{p}}_1,\ldots,\tilde{\boldsymbol{p}}_{t-1})}$, meaning that $p_1$ is a function of $(\tilde{\boldsymbol{p}}_1,\ldots,\tilde{\boldsymbol{p}}_{t-1})$, and we denote by $p_1(\tilde{\boldsymbol{p}}_1,\ldots,\tilde{\boldsymbol{p}}_{t-1})$. Since $\frac{\mathrm{d}}{\mathrm{d}p_1}\left(\frac{h(p_1)}{1-p_1}\right)=\frac{-\log_2(p_1)}{(1-p_1)^2}>0,\forall p_1\in(0,0.5]$, we conclude that $p_1(\tilde{\boldsymbol{p}}_1,\ldots,\tilde{\boldsymbol{p}}_{t-1})$ increases monotonically with $\frac{h(p_1)}{1-p_1}$, thus inversely with $f_t(\tilde{\boldsymbol{p}}_1,\ldots,\tilde{\boldsymbol{p}}_{t-1})$. Therefore, $p_1(\tilde{\boldsymbol{p}}_1,\ldots,\tilde{\boldsymbol{p}}_{t-1})$ is a decreasing function of $f_t(\tilde{\boldsymbol{p}}_1,\ldots,\tilde{\boldsymbol{p}}_{t-1})$.

Among all the tuples of $(\tilde{\boldsymbol{p}}_1, \ldots, \tilde{\boldsymbol{p}}_{t-1})$ that satisfy the constraints of Equation $(2)$, suppose $(\tilde{\boldsymbol{p}}_1^*, \ldots, \tilde{\boldsymbol{p}}_{t-1}^*)$ minimizes $f_t(\tilde{\boldsymbol{p}}_1, \ldots, \tilde{\boldsymbol{p}}_{t-1})$. From the arguments above, we conclude that $(\tilde{\boldsymbol{p}}_1^*, \ldots, \tilde{\boldsymbol{p}}_{t-1}^*)$ together with $p_1(\tilde{\boldsymbol{p}}_1^*, \ldots, \tilde{\boldsymbol{p}}_{t-1}^*) = \frac{1}{f_t(\tilde{\boldsymbol{p}}_1^*, \ldots, \tilde{\boldsymbol{p}}_{t-1}^*)} = \frac{1}{F_t^*}$ achieve the minimum of Problem $(\mathrm{P3})$. This completes the proof. ■

## Acknowledgments

This chapter is in part a reprint of the material in the paper: Lele Wang, Minghai Qin, Eitan Yaakobi, Young-Han Kim, and Paul H. Siegel, "WOM with retained messages", in Proc. IEEE International Symposium of Information Theory (ISIT) 2012, Cambridge, MA, USA, July 2012.

# Chapter 5

# Inter-cell interference free codes for flash memories

## 5.1 Introduction

In Chapter 3 and 4, we studied WOM models that can be applied to flash memory systems where some important physical properties of flash memory cells are not considered. In this chapter, we study constrained codes that take into account the coupling capacitance of adjacent flash memory cells to mitigate the inter-cell interference.

Non-volatile memories — in particular, flash memories — have attracted considerable attention due to their high data-transfer rate and low power consumption [12]. Flash memory cells consist of floating gate transistors, in which the amount of trapped charge determines the cell voltage, referred to as the cell level. A flash memory cell is written to, or "programmed," by applying a suitable voltage to the cell in order to inject the desired amount of charge to reach a certain cell level. Programming precision is an important factor governing the achievable capacity of flash memory storage. Another is inter-cell interference (ICI), caused by the parasitic capacitance between adjacent cells, as a result of which the voltage level of a so-called victim cell may be increased when a high voltage is applied to neighboring cells [21, 58].

As an example of ICI, consider a single-level cell (SLC) flash memory, meaning a memory whose cells supports only two levels. (The SLC designation is somewhat of a misnomer.) We denote the low level by the symbol 0 and the high level by the symbol 1. Now, if a group of three consecutive cells in a row are programmed with the 1 0 1 pattern, the level of the middle cell may be inadvertently increased due to the effect of ICI. During data recovery, the level

of the victim cell may be erroneously interpreted as representing a programmed symbol 1. To combat this effect, the use of a constrained code that prevents the appearance of the ICI-prone symbol pattern 1 0 1 has been proposed. Similar ICI-mitigating constraints for multi-level flash memory cells, preventing the appearance of, say, the pattern $(q-1)\ 0\ (q-1)$ in consecutive $q$-ary cells, have also been considered [5, 59]. We will refer to constrained codes that eliminate these ICI-inducing patterns as ICI-free codes.

In this chapter, we investigate the application of ICI-mitigating codes in two flash memory settings. First, we study information-theoretic limits on the efficiency of binary ICI-free codes that are also balanced, meaning that codewords have an equal number of 0's and 1's. These codes are of interest when a dynamic read threshold is used to reduce the number of errors caused by cell level drift resulting from charge leakage. Specifically, we determine the asymptotic information rate of ICI-free balanced codes.

We then consider codes that allow for the efficient re-use of a binary ICI-free write-once-memory (WOM), that is, a binary WOM that does not support codewords containing the 1 0 1 pattern. The ICI-free WOM provides a model for a flash memory system that uses a multiple-write WOM code to extend the device lifetime, with the added constraint that none of the codewords contain the ICI-inducing 101 pattern. Our main result is a characterization of the $t$-write, sum-rate capacity of an ICI-free WOM, as well as an explicit numerical evaluation for $2 \leqslant t \leqslant 7$.

We now discuss these two applications in more detail.

### 5.1.1 ICI-free Balanced Codes for Dynamic Threshold Detection

Charge leakage, which results in a downward drift in cell voltage levels in a flash memory, can lead to errors in the data retrieval process when the read threshold(s) are fixed. In [97], a dynamic read threshold was proposed as a means to compensate for cell-level drift in SLC flash memory. The threshold adaptation is facilitated by the use of a balanced code, that is, a code in which every codeword has an equal number of 0's and 1's. (See, for example, [52].) (Of necessity, the codeword length is even.)

Under the assumption that the cell-level drift is essentially uniform across all cells, the relative ranking of cell levels is largely preserved. Therefore, by adjusting the threshold value to the point where half of the cell levels are above the threshold and half fall below, the cells programmed to a 1 or a 0 may still be correctly identified with high probability. Since the asymptotic information rate of balanced binary codes is 1, the rate penalty associated with the

use of a balanced code can be made negligibly small with proper code design.

The construction of efficient balanced codes has been extensively studied in [38, 39, 52, 85, 86, 89], and extensions to non-binary and two-dimensional balanced codes have been considered in [62, 63, 67, 83]. Codes that combine the balanced property with certain other constraints, such as runlength limitations, have also been addressed in, for example, [40].

We are specifically interested in binary codes that are both balanced and ICI-free, as defined above. These codes can be used with a dynamic threshold scheme, while also mitigating ICI effects.

### 5.1.2 Coding for an ICI-Free Write-Once-Memory (WOM)

One of the most conspicuous properties of flash memory cells is the asymmetry in the programming process. Cell levels can be easily increased by injecting additional charge into them [12]. In contrast, to decrease the level of even a single cell, the whole block of cells ($\sim 10^6$ cells) containing it has to be erased and then reprogrammed accordingly. These block erasures not only introduce significant latency into the writing process, but also degrade the floating gate cells, thereby shortening the usable lifetime of the device. Therefore, it is desirable to reduce the number of these block erasures in order to enhance the endurance of the flash memory and increase its lifetime storage capacity, which is the total amount of information that can be stored.

The original motivation for the use of rewriting codes came from storage media such as punch cards, optical disks, electronically programmable memories, and paper tapes, all of which consisted of "write-once" bits, or "wits," whose physical states could be changed only once. These technologies could be modeled as a write-once memory (WOM), that is, a binary storage medium consisting of cells supporting two states, designated as 0 and 1, such that during the recording process, a cell can remain in its existing state or, if it is in state 0, can be irreversibly changed to state 1.

Rivest and Shamir [71] showed that, through the use of properly designed codes, a WOM can store multiple generations of information much more efficiently than might have been expected.

Wolf et al. [91] studied binary WOMs from an information-theoretic perspective under various assumptions about state information available at the encoder and decoder. Heegard [36] determined the capacity region of achievable rates for binary WOMs with state information at the encoder, while also introducing several generalized models that allowed for noise, non-binary input and output alphabets, and different types of cell level transitions. The capacity region of

non-binary WOMs with cell-state transitions described by an arbitrary directed acyclic graph was found by Fu and Han Vinck [5]. On the other hand, a WOM can be viewed as a special type of write-efficient memory (WEM) [2]. Within this framework, Fu and Yeung [29] derived the sum-capacity of deterministic WOMs described by a more general graph.

Several other works pertaining to WOMs and WOM codes have appeared, a number of them motivated by the relevance of the WOM model to flash memory devices [27, 30, 43, 51, 87, 88, 93].

In view of the distinct and complementary performance benefits offered by multiple-write WOM codes and ICI-mitigating codes, we investigate their combination in the framework of coding for an input-constrained WOM. Broadly speaking, an input-constrained WOM is a WOM that restricts the input words that it can store by forbidding the appearance of certain symbol patterns. In this framework, an ICI-free WOM is one that does not allow the pattern 1 0 1 to be written at any time.

### 5.1.3 Outline of the Chapter

In Section 5.2.1, we present the derivation of an exact expression for the generating function of the number of binary ICI-free balanced sequences, as well a combinatorial formula for its coefficients. From each of these, we deduce the asymptotic information rate of ICI-free balanced sequences. In Section 5.2.2, we describe a heuristic, probabilistic argument that yields the same rate. Extending it to the non-binary setting, we develop a conjecture for the asymptotic information rate of $q$-ary ICI-free balanced codes. Section 5.3.1 gives background on input-constrained WOMs and two-dimensional constraints. In Section 5.3.2, we prove that the $t$-write sum-capacity of the input-constrained WOM is equal to the capacity of a corresponding two-dimensional constraint defined on an infinite $t$-row strip. Section 5.3.3 describes a construction of 2-write ICI-free WOM codes based upon covering subset partitions of bipartite graphs. Section 5.4 concludes this chapter.

## 5.2 ICI-free Balanced Codes

In this section, we study information-theoretic properties of ICI-free balanced codes. We derive the asymptotic information rate of ICI-free balanced codes over the binary alphabet using combinatorial properties of walks on the two-dimensional lattice $\mathbb{Z}^2$. We then present a heuristic, probabilistic argument that yields the same result. Extending this heuristic argument

to $q$-ary codes, we make a conjecture as to the asymptotic information rate of $q$-ary ICI -free balanced codes.

We begin with some definitions.

**Definition 5.2.1.** *A length-$2n$ binary sequence $u \in \{0,1\}^{2n}$ is said to be **ICI-free balanced** if*

1. *it contains exactly $n$ symbols that are 0 and $n$ symbols that are 1;*
2. *$u_{i-1}, u_i, u_{i+1} \neq 101$, for all $i$ such that $2 \leqslant i \leqslant 2n-1$.*

We refer to a set of ICI-free balanced sequences of the same length as an ICI-free balanced code.

**Definition 5.2.2.** *Let $\mathcal{C}_n$ be the set of all binary ICI-free balanced sequences of length $2n$. The **asymptotic information rate** of $\mathcal{C}_n$ is defined as*

$$C^{\langle 2 \rangle} = \limsup_{n \to \infty} \frac{\log_2 |\mathcal{C}_n|}{2n}.$$

**Remark 5.2.1.** It can be deduced from Lemma 5.2.9 and Lemma 5.2.10 that $\lim_{n\to\infty} \frac{\log_2 |\mathcal{C}_n|}{2n}$ exists. It is also observed by simulation that $\frac{\log_2 |\mathcal{C}_n|}{2n}$ is an increasing sequence and upper bounded by 1.

Referring to Definition 5.2.2, let $C(x)$ be the generating function of $|\mathcal{C}_n|$, that is,

$$C(x) = \sum_{n \geqslant 1} c_n x^n,$$

where $c_n = |\mathcal{C}_n|$ for $n \geqslant 1$. Our main contribution in this section is the following theorem, which gives a closed-form expression for $C(x)$ and determines precisely the asymptotic information rate $C^{\langle 2 \rangle}$.

**Theorem 5.2.3.** *The generating function $C(x)$ is given by*

$$C(x) = \sqrt{\frac{1+x}{1-3x}} - 1,$$

*and the asymptotic information rate of binary ICI-free balanced codes is*

$$C^{\langle 2 \rangle} = \frac{1}{2} \log_2 3.$$

### 5.2.1 Derivation of the Asymptotic Information Rate

This section is devoted to the proof of Theorem 5.2.3.

Let $\mathcal{S}_n$ denote the set of binary balanced sequences of length $2n$. We denote by $\mathcal{A}_n$ the set of all ICI-free balanced sequences of length $2n$ that start with a 1 and by $\mathcal{B}_n$ the set of all ICI-free balanced sequences of length $2n$ that start with a 0. Finally, the cardinalities of these subsets are denoted by $A_n = |\mathcal{A}_n|$ and $B_n = |\mathcal{B}_n|$, and the corresponding generating functions of the cardinalities are denoted by $A(x)$ and $B(x)$, respectively.

The derivation of $C(x)$ will make use of the close connection between ICI-free balanced sequences and paths in the integer lattice $\mathbb{Z}^2$. The following definition introduces several types of paths that will play a role in the analysis.

**Definition 5.2.4.** *A **path** of length $n$ is an $n$-step walk on $\mathbb{Z}^2$, starting at $(0,0)$, such that every step is either an **upstep** obtained by adding $\mathsf{U} = (1,1)$ or a **downstep** obtained by adding $\mathsf{D} = (1,-1)$, respectively, to the current position. If the path is at $(x_i, y_i)$ after $i$ steps, the **height at step** $i$ is defined to be $y_i$.*

*A path of length $n$ is called a **symmetric path** if it ends in $(n,0)$.*

*A path is called a **UDU-free path** if UDU is not a subsequence of the path.*

*A symmetric path is called a **Dyck path** if it never goes below the horizontal axis $y = 0$, i.e, the heights are non-negative after every step.*

Let $\mathcal{P}$ denote the set of all symmetric paths, including the empty path. Let $\mathcal{F}$ be the set of all UDU-free symmetric paths, including the empty path. Let $\mathcal{U} \subset \mathcal{F}$ be the set of non-empty paths that start with a U, let $\mathcal{D} \subset \mathcal{F}$ be the set of non-empty paths that start with a D, and let $\mathcal{H}$ be the set of UDU-free Dyck paths.

Now, let $\mathcal{P}_n \subset \mathcal{P}$ be the set of symmetric paths of length $2n$, for $n \geqslant 0$. Define $\mathcal{F}_n \subset \mathcal{F}$ to be the set of UDU-free paths of length $2n$. We use $F_n = |\mathcal{F}_n|$ to denote the cardinality of $\mathcal{F}_n$, and $F(x)$ to denote the corresponding generating function of $F_n$.

We define in an entirely analogous manner the sets $\mathcal{U}_n, \mathcal{D}_n$, and $\mathcal{H}_n$, their cardinalities $U_n, D_n$, and $H_n$, and their generating functions $U(x), D(x)$ and $H(x)$.

The evident connection between balanced sequences and symmetric paths, as well as between their subsets defined above, is stated formally in the following lemma.

**Lemma 5.2.5.** *There is a bijection between $\mathcal{S}_n$ and $\mathcal{P}_n$. The bijection maps $\mathcal{C}_n$ to $\mathcal{F}_n$ and, more specifically, $\mathcal{A}_n$ to $\mathcal{U}_n$ and $\mathcal{B}_n$ to $\mathcal{D}_n$.*

**Proof.** The bijective mapping between balanced sequences and symmetric paths is obtained by identifying the symbols 1 and 0 with steps U and D, respectively. It is clear that this mapping establishes the bijection between $\mathcal{C}_n$ and $\mathcal{F}_n$, as well as between $\mathcal{A}_n$ and $\mathcal{U}_n$ and between $\mathcal{B}_n$ and $\mathcal{D}_n$. ■

The following lemma follows immediately from the definitions above and the properties of the bijection established in Lemma 5.2.5.

**Lemma 5.2.6.** *The generating functions $C(x)$ and $F(x)$ can be written as:*

$$C(x) = A(x) + B(x) = U(x) + D(x)$$

*and*

$$F(x) = U(x) + D(x) + 1.$$

For any $n \geqslant 1$, we will define a mapping from the subset $\mathcal{U}_n \subset \mathcal{F}_n$ of length-$2n$, UDU-free symmetric paths that begin with U to the subset $\mathcal{D}_n \subset \mathcal{F}_n$ of length-$2n$, UDU-free symmetric paths that begin with D. We will then show that this mapping is actually a bijection. In order to describe the mapping succinctly, we introduce the following terminology and notation.

**Definition 5.2.7.** *Let $u = [u_1, ..., u_{2n-1}, u_{2n}]$ be a path in $\mathcal{U}_n$. The $k$-left cyclic shift of $u$, denoted by $u^{(k)}$, is the path obtained by cyclically shifting $u$ by $k$ steps to the left. That is,*

$$u^{(k)} = [u_{k+1}, \ldots, u_{2n}, u_1, \ldots, u_{k-1}, u_k].$$

Note that all shifts of $u$ are symmetric paths because the number of U steps and D steps remain equal. Define the mapping

$$\phi : \mathcal{U} \mapsto \mathcal{P}$$

as follows: Given a path $u \in \mathcal{U}$, let $i$ be the index of the last symbol D in $u$ such that the path falls from height 1 to height 0. Then $\phi(u) = u^{(i-1)}$, the $(i-1)$-left cyclic shift of $u$.

**Proposition 5.2.8.** *The restriction of the mapping $\phi$ to $\mathcal{U}_n$ is a bijection from $\mathcal{U}_n$ to $\mathcal{D}_n$, for all $n \geqslant 1$. Therefore, $|\mathcal{U}_n| = |\mathcal{D}_n|$, and $U(x) = D(x)$.*

**Proof.** We first show that $\phi(u) \in \mathcal{D}_n, \forall u \in \mathcal{U}_n$.

By construction, $\phi(u)$ is a symmetric path that starts with a D, so we need to show that $\phi(u)$ is UDU-free. Since $u \in \mathcal{U}_n$, this translates into showing that the UDU-free constraint is not violated when $u_1$ (which is a U) is cyclically shifted and concatenated with $u_{2n}$. Let $i$ be the index of the last D that causes the path to fall from height 1 to height 0. If $u_{2n} = \mathsf{D}$, then $i = 2n$

and the first two steps of $\phi(u)$ are $\mathsf{DU}$; otherwise, $u_{2n}u_1 = \mathsf{UU}$. In either case, we conclude that $\phi(u)$ is $\mathsf{UDU}$-free. Hence, the image of $\phi$ lies in $\mathcal{D}_n$.

We now prove the injectivity of $\phi$, that is, if $u, v \in \mathcal{U}_n$ and $u \neq v$, then $\phi(u) \neq \phi(v)$.

For any two distinct paths $u \in \mathcal{U}_n$ and $v \in \mathcal{U}_n$, let $\phi(u) = p$, where $p = [p_1, \ldots, p_{2n}]$ and $\phi(v) = q$, where $q = [q_1, \ldots, q_{2n}]$. Let $i_u$, resp. $i_v$, be the index of the last $\mathsf{D}$ such that $u$, resp. $v$, falls from height 1 to height 0. If $i_u = i_v$, then $p \neq q$ since, by definition of a left cyclic shift, distinct paths that are left-shifted by the same amount must yield distinct paths. On the other hand, suppose $i_u \neq i_v$ and, without loss of generality, assume $i_u < i_v$.

Then we claim that at least one of the following statements is true:

(a) There exists an index $j \in \{1, 2, \ldots, (2n - i_v + 1)\}$ such that $p_j \neq q_j$;

(b) $p_{(2n-i_v+2)} \neq q_{(2n-i_v+2)}$.

Each of these statements implies that $\phi(u) \neq \phi(v)$, as desired.

It suffices to prove that if (a) does not hold, then b) must. To see this, note that $q$ is obtained by the $(i_v - 1)$-left cyclic shift of $v$, implying that $q_{(2n-i_v+1)} = v_{2n}$. Therefore, $q_{(2n-i_v+2)} = v_1 = U$. Meanwhile the height of $q$ after $q_{(2n-i_v+1)}$ is $-1$. Now suppose (a) does not hold, i.e., $[p_1, \ldots, p_{(2n-i_v+1)}] = [q_1, \ldots, q_{(2n-i_v+1)}]$. Then, by construction, $p_{(2n-i_v+1)} = u_{(2n-i_v+i_u)}$ and the height of $u$ after $u_{(2n-i_v+i_u)}$ is 0. Thus, $p_{(2n-i_v+2)} = u_{(2n-i_v+i_u+1)} = D$ since, otherwise, the height of $u$ after $u_{(2n-i_v+i_u+1)}$ is 1 and $u_{i_u}$ would not be the last $\mathsf{D}$ corresponding to a fall from height 1 to height 0 in $u$, contradicting the definition of $i_u$. Thus, $p_{(2n-i_v+2)} = \mathsf{U}$ and $q_{(2n-i_v+2)} = \mathsf{D}$, confirming that condition (b) holds.

This completes the proof that the mapping $\phi$ restricted to $\mathcal{U}_n$ is in injection into $\mathcal{D}_n$, and so $|\mathcal{U}_n| \leqslant |\mathcal{D}_n|$, for all $n \geqslant 1$.

In a similar manner, we define a mapping

$$\gamma : \mathcal{D}_n \to \mathcal{U}_n$$

as follows: Given a path $d \in \mathcal{D}_n$, let $i$ be the label of the first $\mathsf{U}$ such that the path rises from height -1 to height 0. Then $\gamma(d) = d^{(i-1)}$, the $(i-1)$-left cyclic shift of $d$.

The proof that the restriction of the mapping $\gamma$ to $\mathcal{D}_n$ is an injection into $\mathcal{U}_n$, for all $n \geqslant 1$, is similar to that of $\phi$ being an injection, so we omit the details. Consequently, $|\mathcal{D}_n| \leqslant |\mathcal{U}_n|$, for all $n \geqslant 1$.

In fact, one can see that $\gamma$ and $\phi$ are inverse functions of one another, so they in fact define bijections between $\mathcal{U}_n$ and $\mathcal{D}_n$.

We can now conclude that $|\mathcal{U}_n| = |\mathcal{D}_n|$ for all $n \geqslant 1$, and, therefore, $U(x) = D(x)$. ■

**Example 5.2.1.** Figure 5.1 illustrates the bijection between $\mathcal{U}_3$ and $\mathcal{D}_3$. There are 10 UDU-free paths of length 6 and the last D (or first U) that the path falls from height 1 to 0 (or rises from -1 to 0) is circled for each path.

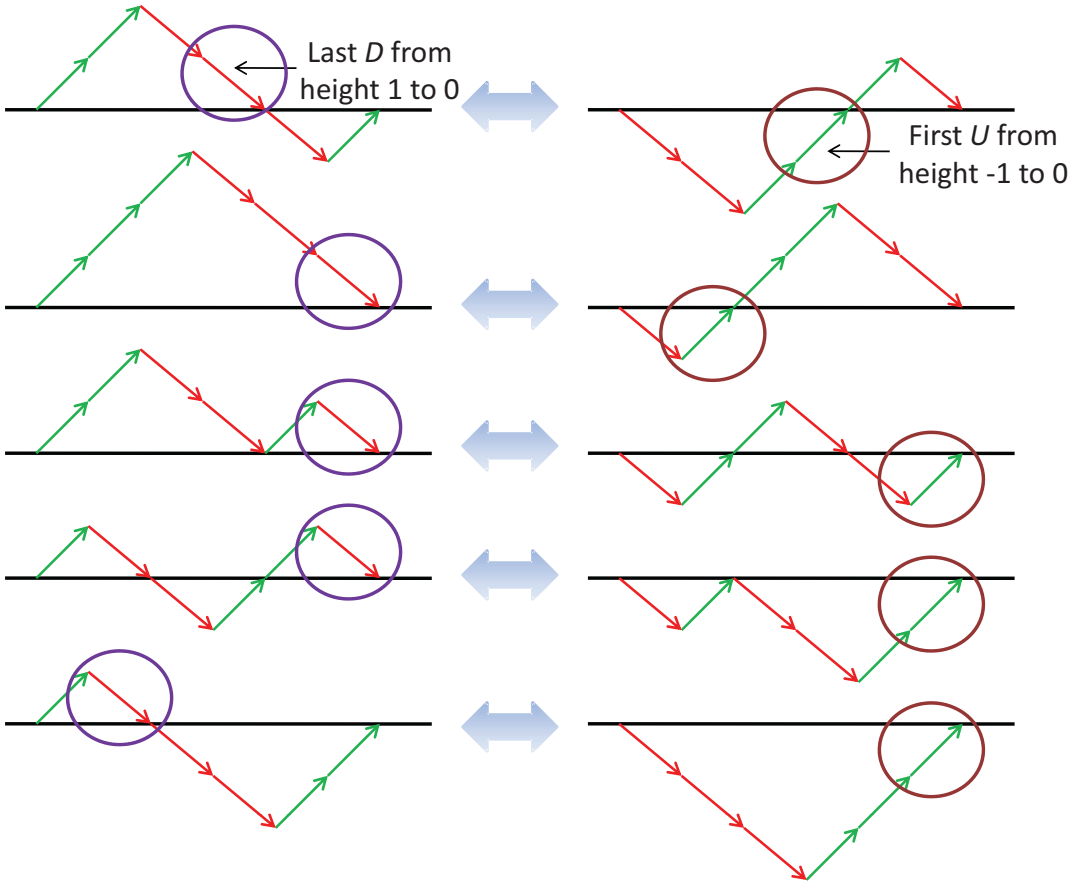


**Figure 5.1**: Bijection between $\mathcal{U}_3$ and $\mathcal{D}_3$

We are now in a position to prove the main result.

Proof of Theorem 5.2.3: Any path in $\mathcal{U}$ can be written in one of the following two forms:

1. UDP, where P is empty or in $\mathcal{D}$; or,
2. UPDQ, where P is a non-empty path in $\mathcal{H}$, and Q is in $\mathcal{F}$.

(This is the so-called first return decomposition of a path in $\mathcal{U}$.)

This gives rise to the equation

$$U(x) = x(1 + D(x)) + x(H(x) - 1)F(x).$$

From Proposition 5.2.8, we have $D(x) = U(x)$. Together with Lemma 5.2.6, this implies that $F(x) = 2U(x) + 1$. Therefore,

$$U(x) = \frac{xH(x)}{1 + x - 2xH(x)}.$$

It was shown in [81] that

$$H(x) = \frac{1 + x - \sqrt{1 - 2x - 3x^2}}{2x}.$$

Therefore,

$$\begin{aligned} U(x) &= \frac{xH(x)}{1 + x - 2xH(x)} \\ &= \frac{1 + x - \sqrt{1 - 2x - 3x^2}}{2\sqrt{1 - 2x - 3x^2}} \\ &= \frac{1}{2}\sqrt{\frac{1 + x}{1 - 3x}} - \frac{1}{2}. \end{aligned}$$

We conclude that

$$C(x) = 2U(x) = \sqrt{\frac{1 + x}{1 - 3x}} - 1.$$

If we treat $C(x)$ as a complex function, the Cauchy-Hadamard Theorem [1, p. 39], [14,33] states that

$$\limsup_{n\to\infty} |c_n|^{1/n} = \frac{1}{\rho},$$

where $\rho$ is the smallest modulus of a singularity of $C(x)$. From the expression for $C(x)$ shown above, we see that

$$\rho = \frac{1}{3}.$$

Recalling that $c_n = |\mathcal{C}_n|$, we conclude that

$$C^{\langle 2\rangle} = \lim_{n\to\infty} \frac{\log |\mathcal{C}_n|}{2n} = \frac{1}{2}\log 3.$$

This completes the proof.

Note that the proof of Theorem 5.2.3 above does not involve explicit expressions for $|U_n|$ and $|D_n|$. However, Deutsch (A005773, [77]) and Callan [11] have shown that

$$U_n = D_n = \sum_{j=0}^{n-1} \binom{j}{\lfloor \frac{j}{2} \rfloor} \binom{n-1}{j} \tag{5.1}$$

and from this formula, one can obtain an alternative derivation of the asymptotic information rate $C^{\langle 2\rangle}$, as we now show.

From Callan's formula, we have

$$\begin{aligned} C^{\langle 2\rangle} &= \lim_{n\to\infty} \frac{\log_2 (U_n + D_n)}{2n} \\ &= \lim_{n\to\infty} \frac{1}{2n} \left( 1 + \log_2 \sum_{j=0}^{n-1} \binom{j}{\lfloor \frac{j}{2} \rfloor} \binom{n-1}{j} \right) \\ &= \lim_{n\to\infty} \frac{1}{2n} \left( \log_2 \sum_{j=0}^{n-1} \binom{j}{\lfloor \frac{j}{2} \rfloor} \binom{n-1}{j} \right). \end{aligned}$$

The following lemma shows that replacing the summation with a maximization does not affect the value of the limit in the formula above.

**Lemma 5.2.9.**

$$C^{\langle 2\rangle} = \lim_{n\to\infty} \frac{1}{2n} \log_2 \max_{0\leqslant j\leqslant n-1} \binom{j}{\lfloor \frac{j}{2} \rfloor} \binom{n-1}{j}.$$

**Proof.** Since all terms in the summation above are positive, we have

$$\begin{aligned} &\lim_{n\to\infty} \frac{1}{2n} \left( \log_2 \sum_{j=0}^{n-1} \binom{j}{\lfloor \frac{j}{2} \rfloor} \binom{n-1}{j} \right) \\ \geqslant &\lim_{n\to\infty} \frac{1}{2n} \log_2 \max_{0\leqslant j\leqslant n-1} \binom{j}{\lfloor \frac{j}{2} \rfloor} \binom{n-1}{j}. \end{aligned}$$

On the other hand, by replacing the sum of terms with their maximum, we have

$$\begin{aligned} &\lim_{n\to\infty} \frac{1}{2n} \left( \log_2 \sum_{j=0}^{n-1} \binom{j}{\lfloor \frac{j}{2} \rfloor} \binom{n-1}{j} \right) \\ \leqslant &\lim_{n\to\infty} \frac{1}{2n} \log_2 \left( n \max_{0\leqslant j\leqslant n-1} \binom{j}{\lfloor \frac{j}{2} \rfloor} \binom{n-1}{j} \right) \\ \leqslant &\lim_{n\to\infty} \frac{\log_2 n}{2n} + \lim_{n\to\infty} \frac{1}{2n} \log_2 \max_{0\leqslant j\leqslant n-1} \binom{j}{\lfloor \frac{j}{2} \rfloor} \binom{n-1}{j} \\ = &\lim_{n\to\infty} \frac{1}{2n} \log_2 \max_{0\leqslant j\leqslant n-1} \binom{j}{\lfloor \frac{j}{2} \rfloor} \binom{n-1}{j} \end{aligned}$$

Therefore,

$$C^{\langle 2\rangle} = \lim_{n\to\infty} \frac{1}{2n} \log_2 \max_{0\leqslant j\leqslant n-1} \binom{j}{\lfloor \frac{j}{2} \rfloor} \binom{n-1}{j}.$$

■

The next lemma gives the limit of the normalized value of argument $j$ that achieves the maximum in Lemma 5.2.9.

**Lemma 5.2.10.**

$$\lim_{n\to\infty} \frac{1}{n} \arg \max_{0\leqslant j\leqslant n-1} \binom{j}{\lfloor \frac{j}{2} \rfloor} \binom{n-1}{j} = \frac{2}{3}.$$

**Proof.** Note that

$$\binom{j}{\lfloor \frac{j}{2} \rfloor} \binom{n-1}{j} = \binom{n-1}{\lfloor \frac{j}{2} \rfloor, \lceil \frac{j}{2} \rceil, n-1-j}.$$

If $n-1$ is a multiple of 3, then this quantity is maximized when $\lfloor \frac{j}{2} \rfloor = \lceil \frac{j}{2} \rceil = n-1-j = \frac{n-1}{3}$, i.e., when $j = \frac{2(n-1)}{3}$. Similar reasoning shows that the maximizing values of $j$ for all $n \geqslant 1$ satisfy $\lim_{n\to\infty} \frac{j}{n} = \frac{2}{3}$. ■

From Lemma 5.2.9 and Lemma 5.2.10, we conclude that

$$C^{\langle 2 \rangle} = \frac{1}{2} \log_2 3.$$

**Remark 5.2.2.** The capacity of the "no 101" constraint is approximately 0.8114. Compared to $C_1$, there is a 2% rate loss due to the additional balanced constraint. Balanced codes are special types of constant weight codes [9], where the weight is half of the code length $n$. In general, constant weight codes can be used to adapt to the voltage drift during read cycles as well. It might be better to use constant weight codes with weight less than $\frac{n}{2}$ if rate optimization is the only figure of merit; however, balanced codes have the advantage of easy encoding and decoding, while sacrificing only a small portion (2%) of the rate.

### 5.2.2 Heuristic Probabilistic Derivation

We now provide a heuristic probabilistic argument that yields the result derived in the previous section, namely $C^{\langle 2 \rangle} = \frac{1}{2} \log_2 3$. Extending this argument to $q$-ary ICI-free balanced codes, we arrive at a conjecture for their asymptotic information rate $C^{\langle q \rangle}$.

Let $\mathcal{S}$ be the set of all balanced sequences of length $2n$. It is evident that $|\mathcal{S}| = \binom{2n}{n}$. Now, let $Z$ be a randomly chosen sequence in $\mathcal{S}$, and let $Z_i$ be the $i$-th entry in $Z$, for $1 \leqslant i \leqslant 2n$. Clearly $\mathbb{P}(Z_i = 1) = \mathbb{P}(Z_i = 0) = \frac{1}{2}$ and, if $n$ is sufficiently large, $\mathbb{P}(Z_{i-1} = Z_{i+1} = 1 | Z_i = 0)$ is approximately $\frac{1}{4}$. Define a sequence of events $\mathtt{E}_i \stackrel{\text{def}}{=} \{(Z_{i-1}, Z_i, Z_{i+1}) \neq (101) | Z_i = 0\}$, for $2 \leqslant i \leqslant 2n-1$. Then $\mathbb{P}(\mathtt{E}_i) \approx \frac{3}{4}$. The number of 0's in $Z$ is $n$, so if we treat the events $\mathtt{E}_i$ as independent (though, in reality, they are not), then the probability that $Z$ satisfies the ICI-free

balanced constraint is approximately $\left(\frac{3}{4}\right)^n$. Thus, the number of ICI-free balanced sequences in $\mathcal{S}$ is approximately $\binom{2n}{n}\left(\frac{3}{4}\right)^n$. That is,

$$\frac{\log_2|\mathcal{C}_n|}{2n} \approx \frac{\log_2\left(\binom{2n}{n}\left(\frac{3}{4}\right)^n\right)}{2n} \xrightarrow{n\to\infty} \frac{1}{2}\log_2 3.$$

Although the independence assumption is not valid, this line of reasoning yields the correct answer because the dependency of $\mathtt{E}_i$ and $\mathtt{E}_j$ decreases as $|i-j|$ increases.

Now, recall that, for $q$-level flash cells, ICI arises when three consecutive cells are programmed to the levels $(c_1, c_2, c_3)$ such that $c_1$ and $c_3$ are much larger than $c_2$. It is expected that the most severe ICI will occur when three consecutive cells are programmed to the levels $((q-1), 0, (q-1))$. We now extend the definition of ICI-free balanced sequences to the $q$-ary case to avoid the most severe ICI pattern.

**Definition 5.2.11.** *A $q$-ary sequence $u \in \{0, 1, \ldots, q-1\}^{qn}$ is said to satisfy the $q$-ary ICI-free balanced constraint if*

1. *$\forall j$, such that $0 \leqslant j \leqslant q-1$, the number of $j$'s in $u$ is $n$;*
2. *$(u_{i-1}, u_i, u_{i+1}) \neq ((q-1), 0, (q-1))$, for all $i$ such that $2 \leqslant i \leqslant qn-1$.*

Let $\mathcal{C}_n^{\langle q\rangle}$ be the set of all $q$-ary ICI-free-balanced sequences of length $qn$. The asymptotic information rate of $\mathcal{C}_n^{\langle q\rangle}$ is defined as

$$\mathrm{C}^{\langle q\rangle} = \lim_{n\to\infty} \frac{\log_2|\mathcal{C}_n^{\langle q\rangle}|}{nq}.$$

By direct analogy to the heuristic argument used in the binary case, $q = 2$, one might conjecture that the asymptotic information rate is

$$\mathcal{C}_{\text{conj}}^{\langle q\rangle} = \log_2 q + \frac{1}{q}\log_2\left(\frac{q^2-1}{q}\right).$$

However, based on [73], $\mathcal{C}^{\langle 3\rangle} \approx 1.5258$ while the heuristic argument overestimates $\mathcal{C}^{\langle 3\rangle}$, yielding $\mathcal{C}_{\text{conj}}^{\langle 3\rangle} \approx 1.5283$.

## 5.3 ICI-free WOM Codes

In this section, we study the WOM model with certain input constraints. We first present the definition of an input-constrained WOM and then provide a derivation of the $t$-write sum-capacity. Finally, we give code constructions based on coverings of bipartite graphs.

### 5.3.1 Definitions

Suppose the number of cells is $n$ and the number of rewriting cycles is $t$. The cell levels of a generalized $q$-level WOM after the $i$-th write are denoted by $y_{i,1}^n \in [0:q-1]^n$, for $i \in [1:t]$, where $[k_1:k_2] \stackrel{\text{def}}{=} \{k \in \mathbb{Z} | k_1 \leqslant k \leqslant k_2\}$ and $y_{i,k_1}^{k_2} \stackrel{\text{def}}{=} (y_{i,k_1}, y_{i,k_1+1}, \ldots, y_{i,k_2})$. We will use $[n]$ as a shorthand for $[1:n]$ when no confusion could occur. Furthermore, we write $x_1^n \succeq y_1^n$ if and only if $\forall i \in [n], x_i \geqslant y_i$. We can describe the discrete memoryless generalized WOM by a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of vertices and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges. For a $q$-level WOM, $\mathcal{V} = \{0, 1, \ldots, q-1\}$. If the level can be changed directly from $s_1$ to $s_2$, where $s_1, s_2 \in \mathcal{V}$, then there exists an edge from $s_1$ to $s_2$ and we denote it by $(s_1, s_2) \in \mathcal{E}$. For a state sequence $(s_0, s_1, \ldots, s_t)$, if $\forall i \in [0:t-1], (s_i, s_{i+1}) \in \mathcal{E}$, then we say the path $(s_0, s_1, \ldots, s_t)$ exists and we denote it by $s_0 \to s_1 \to \cdots \to s_t$. For two vectors $(x_{1,1}^n, x_{2,1}^n) \in [0:q-1]^n \times [0:q-1]^n$, we write $x_{1,1}^n \Rightarrow x_{2,1}^n$ if and only if $\forall i \in [n]$, $(x_{1,i}, x_{2,i}) \in \mathcal{E}$. The transition matrix $A = \left(a_{i,j}\right) \in \{0,1\}^{q \times q}$ is defined as follows. For $i, j \in [0:q-1]$, $a_{ij} = 1$ if $(i, j) \in \mathcal{E}$; otherwise, $a_{ij} = 0$.

**Definition 5.3.1.** *Let $y_{i,1}^n$ denote the cell-state vector after the $i$-th write, for $i \in [t]$. An $[n, t; 2^{nR_1}, \ldots, 2^{nR_t}]$ $q$-ary WOM code $\mathcal{C}_{q,\mathcal{G}}$ described by the graph $\mathcal{G}$ is a coding scheme consisting of $n$ cells and $t$ pairs of encoders and decoders $(\mathcal{E}_i, \mathcal{D}_i)$, where $\forall i \in [t]$, the encoder is a mapping*

$$\mathcal{E}_i : [1 : 2^{nR_i}] \times Im\{\mathcal{E}_{i-1}\} \to [0:q-1]^n,$$

*such that $\forall (m, y_{i-1,1}^n) \in [1 : 2^{nR_i}] \times Im\{\mathcal{E}_{i-1}\}$,*

$$y_{i-1,1}^n \Rightarrow y_{i,1}^n = \mathcal{E}_i(m, y_{i-1,1}^n),$$

*where, by abuse of notation, we use $Im\{\mathcal{E}_0\}$ to represent the initial cell-state vector $\{(0, ..., 0)\}$. The decoder is a mapping*

$$\mathcal{D}_i : Im\{\mathcal{E}_i\} \to [1 : 2^{nR_i}],$$

*such that $\forall m \in [1 : 2^{nR_i}]$,*

$$\mathcal{D}_i(\mathcal{E}_i(m, y_{i-1,1}^n)) = m.$$

*Let $w$ be a $q$-ary sequence. An* **input-constrained WOM code $\mathcal{C}_{S_w,q,\mathcal{G}}$ avoiding** *$w$ is a $q$-ary WOM code such that $w$ is not a subsequence of any codeword in $\mathcal{C}_{S_w,q,\mathcal{G}}$.*

**Definition 5.3.2.** *A rate tuple $(R_1, \ldots, R_t)$ is said to be* achievable *if there exists a sequence of $[n, t; 2^{nR_1}, \ldots, 2^{nR_t}]$ WOM codes. The capacity region is defined as the closure of the set of*

*all achievable rate tuples. The sum-capacity is defined as the supremum of achievable sum-rates* $\sum_{i=1}^{t} R_i$.

To mitigate ICI, the sequence is chosen such that $w \stackrel{\text{def}}{=} (101)$ and the corresponding input-constrained WOM code is called the **binary ICI-free WOM code**. In this section, we are interested in the sum-capacity of the ICI-free binary WOM, i.e., the supremum sum-rate of $\mathcal{C}_{S_{101},2,\mathcal{G}}$, where $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $\mathcal{V} = \{0,1\}$, $\mathcal{E} = \{(0,0),(0,1),(1,1)\}$, and, therefore, $A = \left[\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right]$. We will first provide general results for an arbitrary constraint $S$ and arbitrary number of levels $q$, and then apply them in the binary ICI-free WOM setting.

There is a connection between two-dimensional constrained codes and codes for input-constrained WOMs. Specifically, every $t$-write WOM code of length $n$ can be expressed as a set of $t \times n$ arrays where the $i$-th row, $i \in [t]$, corresponds to the memory state after the $i$-th write. We will exploit this fact in our derivation of the $t$-write sum-capacity. Let us first recall the definition of the capacity of a two-dimensional constraint.

**Definition 5.3.3.** *Given a two-dimensional constraint* $S^{2D}$*, its capacity is defined to be*

$$C_{2D}(S^{2D}) = \lim_{m,n\to\infty} \frac{\log_2 N_S(m,n)}{mn},$$

*where* $N_S(m,n)$ *is the number of* $m \times n$ *arrays that satisfy the constraint* $S^{2D}$*. The* $t$***-write column capacity*** *is defined to be*

$$C(t, S^{2D}) = \lim_{n\to\infty} \frac{\log_2 N_S(t,n)}{n}.$$

**Remark 5.3.1.** The exact capacity of most non-trivial two-dimensional constraints is not known. However, the $t$-write column capacity can be calculated numerically with the aid of the characteristic function of the adjacency matrix associated with the constraint $S^{2D}$ when we fix one dimension of the 2-D array to be of size $t$ [57].

There are a number of two-dimensional constraints that have been extensively studied, e.g., 2-D $(d,k)$-runlength-limited (RLL) [50], no isolated bits [23,34], and checkerboard [65,72]. For the input-constrained WOM codes $\mathcal{C}_{S_w,q,\mathcal{G}}$, where $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we define a constraint on two-dimensional arrays, denoted by $S_w^{2D}$, that is used to calculate the sum-capacity. Specifically, in a $q$-ary two dimensional array $B = \{b_{i,j}\}_{m\times n}$, we must have

$$(b_{i,j}, b_{i+1,j}) \in \mathcal{E}, \forall i \in [m-1], j \in [n]$$

and the pattern $w$ must not be a subsequence in any row of $B$. However, we denote the two-dimensional constraint corresponding to the ICI-free WOM by $S_{101}^{2D}$.

### 5.3.2 Sum-Capacity

The following theorem characterizes the $t$-write sum-capacity of an input-constrained WOM in terms of the $t$-write column capacity of an associated two-dimensional constraint.

**Theorem 5.3.4.** *The $t$-write sum-capacity of the $q$-ary input-constrained WOM that does not allow $w$ is $C(t, S_w^{2D})$.*

*In particular, for the binary ICI-free WOM, the $t$-write sum-capacity is $C(t, S_{101}^{2D})$.*

In order to prove Theorem 5.3.4, we will make use of the characterization of the sum-capacity of a WOM described by a general directed graph $\mathcal{G}$, as presented in [29, Prop. 2]. The derivation in [29] uses results about the sum-capacity of write-efficient memories [2]. Here we will present an alternative derivation based upon the Markov-chain WOM model from [36].

**Lemma 5.3.5.** *The sum-capacity $C_{sum}(t, \mathcal{G})$ of the generalized discrete memoryless $q$-level WOM described by graph $\mathcal{G}$ is the base-2 logarithm of the number of length-$t$ paths that starts from state 0, i.e.,*

$$C_{sum}(t, \mathcal{G}) = \log_2(\boldsymbol{\delta}_{0,q}^T \cdot A_{\mathcal{G}}^t \cdot \mathbf{1}_q),$$

*where $A_{\mathcal{G}}$ is the transition matrix of the graph $\mathcal{G}$, and $\boldsymbol{\delta}_{i,q} = (0, \ldots, 0, 1, 0, \ldots, 0)^T, 0 \leqslant i \leqslant q-1$, is a column vector of length $q$ such that the $(i+1)$-st entry is 1 and the remaining entries are 0's.*

**Proof.** We adopt the notation used in [36]. According to Theorem 3 in [36], the sum-capacity equals

$$\begin{aligned} C_{\text{sum}} &= \sum_{i=1}^{t} R_i \\ &= \sum_{i=1}^{t} H(Y_i|Y_{i-1}) \\ &= H(Y_1, Y_2, \ldots, Y_t), \end{aligned}$$

where $Y_i, i \in [t]$ is a random variable representing the state of the WOM after the $i$-th write. The last equality follows from the fact that $Y_1 \to Y_2 \to \cdots \to Y_t$ form a Markov chain.

First we prove $C_{\text{sum}} \leqslant \log_2(\boldsymbol{\delta}_{0,q}^T \cdot A^t \cdot \mathbf{1}_q)$. The random vector $Y_1^t = (Y_1, \ldots, Y_t)$ corresponds to a path in $\mathcal{G}$; thus, the cardinality of $Y_1^t$ is upper bounded by the number of length-$t$ paths from state 0. Therefore, $H(Y_1^t) \leqslant \log_2(\boldsymbol{\delta}_{0,q}^T \cdot A^t \cdot \mathbf{1}_q)$.

Next we prove achievability. Let $p(y_1^t)$ be the joint probability mass function of $Y_1^t$. Let $p(y_1^t)$ be factored as $p_1(y_1)p_2(y_2|y_1)\cdots p_t(y_t|y_{t-1})$, where $p_i(y_i|y_{i-1})$ is the conditional

transition probability for the $i$-th write. We show that by appropriately choosing $p_1(y_1)p_2(y_2|y_1)$ $\cdots p_t(y_t|y_{t-1})$, $(Y_1, \ldots, Y_t)$ is uniformly distributed on its support. Let

$$p_1(y_1 = j) = \begin{cases} \frac{\boldsymbol{\delta}_{j,q}^T \cdot A^{t-1} \cdot \mathbf{1}_q}{\boldsymbol{\delta}_{0,q}^T \cdot A^t \cdot \mathbf{1}_q}, \text{ if } (0, j) \in \mathcal{E}; \\ 0, \text{ otherwise,} \end{cases}$$

for all $j \in [0 : q-1]$. For $2 \leqslant i \leqslant t$, let

$$p_i(y_i = j|y_{i-1} = \ell) = \begin{cases} \frac{\boldsymbol{\delta}_{j,q}^T \cdot A^{t-i} \cdot \mathbf{1}_q}{\boldsymbol{\delta}_{\ell,q}^T \cdot A^{t-i+1} \cdot \mathbf{1}_q}, \text{ if } (\ell, j) \in \mathcal{E}, \{A^{i-1}\}_{0,\ell} = 1; \\ 0, \text{ otherwise,} \end{cases}$$

for all $j, \ell \in [0 : q-1]$.

Then, for a state sequence $(s_1, s_2, \ldots, s_t)$, we have

$$\begin{aligned} & P(Y_1 = s_1, Y_2 = s_2, \ldots, Y_t = s_t) \\ =& p_1(y_1 = s_1)p_2(y_2 = s_2|y_1 = s_1) \\ & \quad \cdot p_3(y_3 = s_3|y_2 = s_2) \cdots p_t(y_t = s_t|y_{t-1} = s_{t-1}) \\ =& \frac{\boldsymbol{\delta}_{s_1,q}^T \cdot A^{t-1} \cdot \mathbf{1}_q}{\boldsymbol{\delta}_{0,q}^T \cdot A^t \cdot \mathbf{1}_q} \cdot \frac{\boldsymbol{\delta}_{s_2,q}^T \cdot A^{t-2} \cdot \mathbf{1}_q}{\boldsymbol{\delta}_{s_1,q}^T \cdot A^{t-1} \cdot \mathbf{1}_q} \\ & \quad \cdot \frac{\boldsymbol{\delta}_{s_3,q}^T \cdot A^{t-3} \cdot \mathbf{1}_q}{\boldsymbol{\delta}_{s_2,q}^T \cdot A^{t-2} \cdot \mathbf{1}_q} \cdots \frac{\boldsymbol{\delta}_{s_t,q}^T \cdot A^0 \cdot \mathbf{1}_q}{\boldsymbol{\delta}_{s_{t-1},q}^T \cdot A \cdot \mathbf{1}_q} \\ =& \frac{1}{\boldsymbol{\delta}_{0,q}^T \cdot A^t \cdot \mathbf{1}_q} \end{aligned}$$

if the path $s_1 \to \cdots \to s_t$ exists; otherwise,

$$P(Y_1 = s_1, Y_2 = s_2, \ldots, Y_t = s_t) = 0.$$

This proves that $(Y_1, \ldots, Y_t)$ is uniformly distributed on its support set. Since the cardinality of the support set is $\boldsymbol{\delta}_{0,q}^T \cdot A^t \cdot \mathbf{1}_q$, then $H(Y_1, \ldots, Y_t) = \log_2(\boldsymbol{\delta}_{0,q}^T \cdot A^t \cdot \mathbf{1}_q)$. ■

**Example 5.3.1.**

For the state transition diagram in Figure 5.2, suppose the number of writes is $t = 4$. We set the conditional probabilities as follows:

$$p_1(y_1 = 0) = \frac{7}{11}, p_1(y_1 = 1) = \frac{4}{11}.$$

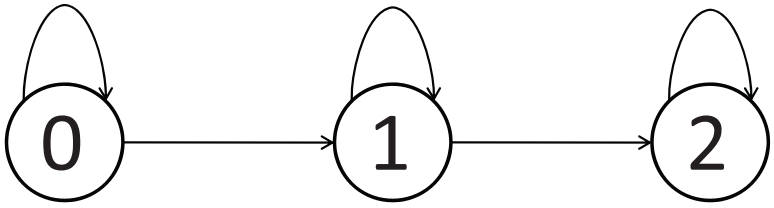


**Figure 5.2**: Generalized WOM with state transition diagram

$$p_2(y_2 = 0|y_1 = 0) = \frac{4}{7}, p_2(y_2 = 1|y_1 = 0) = \frac{3}{7}.$$
$$p_2(y_2 = 1|y_1 = 1) = \frac{3}{4}, p_2(y_2 = 2|y_1 = 1) = \frac{1}{4}.$$
$$p_3(y_3 = 0|y_2 = 0) = \frac{2}{4}, p_3(y_3 = 1|y_2 = 0) = \frac{2}{4}.$$
$$p_3(y_3 = 1|y_2 = 1) = \frac{2}{3}, p_3(y_3 = 2|y_2 = 1) = \frac{1}{3}.$$
$$p_3(y_3 = 2|y_2 = 2) = 1.$$
$$p_3(y_4 = 0|y_3 = 0) = \frac{1}{2}, p_3(y_4 = 1|y_3 = 0) = \frac{1}{2}.$$
$$p_3(y_4 = 1|y_3 = 1) = \frac{1}{2}, p_3(y_4 = 2|y_3 = 1) = \frac{1}{2}.$$
$$p_3(y_4 = 2|y_3 = 2) = 1.$$

Then, each possible state sequence has probability $\frac{1}{11}$, which means the 4-write sum-capacity is $\log_2 11$.

Now we are ready to prove Theorem 5.3.4. We give the proof for the case of a binary ICI-free WOM, i.e., the transition diagram $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined by $\mathcal{V} = \{0, 1\}$ and $\mathcal{E} = \{(0, 0), (0, 1), (1, 1)\}$, and the input constraint is given by $w = (101)$. The generalization to an arbitrary input-constrained WOM follows a similar line of reasoning. **Proof of Theorem 5.3.4.**

Proof of achievability:

Let $n$ and $m$ be two positive integers such that $n$ is a multiple of $(m + 2)$, i.e., $n = \ell(m + 2)$. The memory consists of $n$ cells, denoted by $(c_1, \ldots, c_n)$, which are partitioned into $\ell$ blocks, each with $(m + 2)$ cells. When the messages are written into the memory, within each block, the last 2 cells are kept at level 0, i.e., $c_{i(m+2)} = c_{i(m+2)-1} = 0, \forall i \in [\ell]$. In this way, it can be guaranteed that no 3 consecutive cells at the boundaries of adjacent blocks are $(101)$. Each block of $m$ cells constitutes the same $t$-write WOM code that avoids 101. To be more precise, we first introduce the following definitions.

Let $b : \mathbb{Z}_+ \mapsto \{0,1\}^m$ be the function that maps a non-negative integer $M \in [0, 2^m - 1]$ to its binary representation of length $m$, and let $b^{-1}(\boldsymbol{x})$ be the inverse function for $\boldsymbol{x} \in \{0,1\}^m$.

The following construction yields a sequence of binary ICI-free WOM codes with the claimed sum-rate efficiency.

**Construction 5.3.1** Let $n$, $m$ and $\ell$ be positive integers such that $n = \ell(m+2)$. Suppose the cell-state vector is $c_{i,1}^n \in \{0,1\}^n$ after the $i$-th write, for $i \in [t]$. Let $y_{i,1}^\ell \in [0 : 2^m - 1]^\ell$ satisfy $y_{i,j} = b^{-1}(c_{i,(j-1)(m+2)+1}^{(j-1)(m+2)+m})$, for $j \in [\ell]$.

A directed graph $\mathcal{G}_m = (\mathcal{V}, \mathcal{E})$ with $2^m$ vertices/states is defined as follows. The vertex set is $\mathcal{V} = [0 : 2^m - 1]$, and $\forall i, j \in \mathcal{V}$, $(i,j) \in \mathcal{E} \Leftrightarrow b(j) \succeq b(i)$ and $(101)$ is not a subsequence of $b(j)$. Let $A_m$ be the transition matrix for $\mathcal{G}_m$.

Let $\mathcal{C}_W$ be an $[\ell, t; 2^{\ell R_1}, 2^{\ell R_2}, \ldots, 2^{\ell R_t}]$ $t$-write $2^m$-ary WOM code of length $\ell$ described by $\mathcal{G}_m$. Let $\mathcal{E}_i(m_i, y_{i-1,1}^\ell)$ be its encoder on the $i$-th write, for $i \in [t]$. An $[n, t; 2^{\ell R_1}, \ldots, 2^{\ell R_t}]$ binary ICI-free WOM code $\mathcal{C}_{\text{ICI}}$ of length $n$ is constructed as follows. On the $i$-th write, the encoder uses the following rules:

1. in each block of size $(m+2)$, the last two cells are kept as 0, i.e.,

$$c_{i,(m+2)j-1}^{(m+2)j} = c_{i-1,(m+2)j-1}^{(m+2)j} = 0, \forall j \in [\ell].$$

2. write the message $M_i \in [1 : 2^{\ell R_i}]$ to the remaining $m\ell$ cells. Specifically, let $y_{i,1}^\ell = \mathcal{E}_i(M_i, y_{i-1,1}^\ell)$, and write the remaining $m\ell$ cells such that $c_{i,(j-1)(m+2)+1}^{(j-1)(m+2)+m} = b(y_{i,j}), \forall j \in [\ell]$.

The decoder can be designed accordingly and we omit the details. □

If $\mathcal{C}_W$ is sum-rate optimal, then the sum-rate of $\mathcal{C}_{\text{ICI}}$ is

$$\begin{aligned} R_{m,\text{ICI}}(t) &= \frac{\ell \sum_{i=1}^t R_i}{\ell(m+2)} \\ &= \frac{C_{\text{sum}}(t, \mathcal{G}_m)}{m+2} \\ &= \frac{\log_2(\boldsymbol{\delta}_{0,q}^T \cdot A_m^t \cdot \mathbf{1}_q)}{m+2} \\ &= \frac{\log_2(\boldsymbol{\delta}_{0,q}^T \cdot A_m^t \cdot \mathbf{1}_q)}{m} \cdot \frac{m}{m+2}. \end{aligned}$$

Note that $\boldsymbol{\delta}_{0,q}^T A_m^t \mathbf{1}_q$ counts the number of binary arrays $B = \{b_{i,j}\}_{t \times m}$ such that $b_{i+1,j} \geqslant b_{i,j}, \forall i \in [t-1], j \in [m]$, and the pattern $(101)$ is not a subsequence of any row in $B$.

Letting $m$ go to infinity, we see that there exists a sequence of $t$-write ICI-free WOM codes with rates

$$\begin{aligned} R_{\mathrm{ICI}}(t) &= \lim_{m\to\infty} R_{m,\mathrm{ICI}}(t) \\ &\geqslant \lim_{m\to\infty} \frac{\log_2(\boldsymbol{\delta}_{0,q}^T \cdot A_m^t \cdot \mathbf{1}_q)}{m} \cdot \frac{m}{m+2} \\ &= C(t, S_{101}^{2D}). \end{aligned}$$

The existence of the limit can be shown by the sub-additive property [61] of binary arrays $B$.

Proof of converse:

The converse can be easily proved by noting that if there exists a genie that, at decoding step $j \in [t]$, can provide all of the sequences written into the WOM from the first to the $(j-1)$-st write, then the sum-capacity equals the $t$-write column capacity, which is $C(t, S_{101}^{2D})$. However, this genie does not exist, so for any $t$-write ICI-free WOM code with sum-rate $R_{\mathrm{ICI}}(t)$, it follows that

$$R_{\mathrm{ICI}}(t) \leqslant C(t, S_{101}^{2D}).$$

■

**Example 5.3.2.** Table 5.1 shows the $t$-write sum-capacity $C_{\mathrm{ICI}}(t)$ of the ICI-free WOM, calculated using the techniques in [90]. Also shown is the $t$-write capacity of an unconstrained WOM, which is $\log(t+1)$, for $2 \leqslant t \leqslant 7$. An interesting observation is that $\frac{C_{\mathrm{ICI}}}{\log(t+1)}$ is close to 0.79.

| $t$ | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| $C_{\mathrm{ICI}}(t)$ | 1.264 | 1.584 | 1.831 | 2.035 | 2.207 | 2.356 |
| $\log(t+1)$ | 1.585 | 2 | 2.322 | 2.585 | 2.807 | 3 |

**Table 5.1**: Sum-capacity of ICI-free WOM

### 5.3.3 Code Constructions

We now proceed to the construction of some input-constrained WOM codes. In particular, we construct binary ICI-free WOM codes for $t = 2$ writes. The construction technique generalizes to $q$-ary alphabets with $q > 2$, $t > 2$ writes, and more general input constraints.

Let $\mathcal{C}$ be the set of binary vectors of length $n$ that avoid 101. Let $\mathcal{C} = \mathcal{L} \cup \mathcal{R}$ be a partition of $\mathcal{C}$. For a pair of vectors $(\ell, r) \in \mathcal{L} \times \mathcal{R}$, we say $r$ covers $\ell$ if $r \succeq \ell$. A bipartite graph

$\mathcal{B} = (\mathcal{L} \cup \mathcal{R}, \mathcal{E})$ is defined where $\mathcal{L}$ and $\mathcal{R}$ are the sets of left and right nodes, respectively. An edge $(\ell, r)$ connecting $\ell \in \mathcal{L}$ and $r \in \mathcal{R}$ exists if $r$ covers $\ell$ and we denote such an edge by $(\ell, r) \in \mathcal{E}$. For $\hat{\mathcal{R}} \subseteq \mathcal{R}$, the **covering** of $\hat{\mathcal{R}}$, denoted by $\mathrm{CV}(\hat{\mathcal{R}})$, is defined as $\{\ell \in \mathcal{L} : \exists r \in \hat{\mathcal{R}}, r \text{ covers } \ell\}$ and the **covering cardinality** of $\hat{\mathcal{R}}$ is defined as $|\mathrm{CV}(\hat{\mathcal{R}})|$. We say that $\widetilde{\mathcal{R}} \subseteq \mathcal{R}$ is a **covering subset** if $\mathrm{CV}(\widetilde{\mathcal{R}}) = \mathcal{L}$. A partition $\mathcal{R} = \cup_{i=1}^{k} \mathcal{R}_i$ is called a **covering subset partition** of $\mathcal{R}$ if $\mathcal{R}_i$ is a covering subset for all $i \in [k]$.

**Lemma 5.3.6.** *Let $\mathcal{R} = \cup_{i=1}^{k} \mathcal{R}_i$ be a covering subset partition. Then there exists a 2-write ICI-free WOM code of length $n$ with rate pair $(\frac{\log|\mathcal{L}|}{n}, \frac{\log k}{n})$.*

For a bipartite graph, finding the maximum covering subset partition $k$ is an interesting problem in its own right. In [92], a greedy algorithm is proposed to find covering subset partitions. We extend the greedy algorithm in [92] by adding another parameter $g$ that controls the level of greediness. The algorithm in [92] would coincide with the following algorithm for $g = 1$.

**Algorithm 5.3.7.** FINDING COVERING SUBSETS OF A BIPARTITE GRAPH

**Input**:

a bipartite graph $\mathcal{B} = (\mathcal{L} \cup \mathcal{R}, \mathcal{E})$, where $\mathcal{L} = \{\ell_1, \ldots, \ell_n\}$ $\mathcal{R} = \{r_1, \ldots, r_m\}$;

a positive integer $g$ that measure the extent of greediness in searching for a covering subset;

**Output**:

a partition of $\mathcal{R} = \cup_{i=1}^{k} \mathcal{R}_i$ such that $\mathcal{R}_i$ is a covering subset for all $i \in [k]$.

1: $k \leftarrow 0$;

2: $\mathcal{R}_{\text{unused}} \leftarrow \mathcal{R}$;

3: Mark all $\ell_i \in \mathcal{L}, i \in [n]$ as "uncovered";

4: $\mathcal{R}_{\text{temp}} \leftarrow \emptyset$;

5: **if** $\mathcal{R}_{\text{unused}} = \emptyset$

6: **return** $(\mathcal{R}_1, \ldots, \mathcal{R}_{k-1}, \mathcal{R}_k \cup \mathcal{R}_{\text{temp}})$;

7: **end if**

8: Choose $\hat{\mathcal{R}} \subseteq \mathcal{R}_{\text{unused}}$ such that $|\hat{\mathcal{R}}| \leqslant g$ and $\hat{\mathcal{R}}$ has the largest covering cardinality $|\mathrm{CV}(\hat{\mathcal{R}})|$; /* In case of a tie, choose $\hat{\mathcal{R}}$ with minimum cardinality $|\hat{\mathcal{R}}|$; if there is still a tie, choose any.*/

9: $\mathcal{R}_{\text{temp}} \leftarrow \mathcal{R}_{\text{temp}} \cup \hat{\mathcal{R}}$;

10: $\mathcal{R}_{\text{unused}} \leftarrow \mathcal{R}_{\text{unused}} \setminus \hat{\mathcal{R}}$;

11: Mark $\ell_i \in \mathrm{CV}(\hat{\mathcal{R}})$ as "covered";
12: **If** for all $i \in [n]$, $\ell_i$ are covered,
13: $k \leftarrow k+1$;
14: $\mathcal{R}_k \leftarrow \mathcal{R}_{\text{temp}}$;
15: Go to Step 3;
16: **else**
17: Go to Step 5;
18: **end if**

The following construction uses Algorithm 5.3.7 to construct a two-write ICI-free WOM code.

**Construction 5.3.2** Let $m, n, n', \ell$ be integers such that $m \leqslant n$ and $n' = (n+1)\ell$. Let $\mathcal{C}_n$ be the set of binary vectors that avoid 101 of length $n$, and let $\mathcal{C}_n = \mathcal{L} \cup \mathcal{R}$ be a partition of $\mathcal{C}_n$ such that $\mathcal{L} = \{\boldsymbol{x} \in \mathcal{C}_n : \mathrm{wt}(\boldsymbol{x}) \leqslant m\}$ and $\mathcal{R} = \mathcal{C}_n \setminus \mathcal{L}$. Let $M_1 = |\mathcal{L}|$ and $f_1 : [0 : M_1 - 1] \to \mathcal{L}$ be an arbitrary bijective function. Let $\mathcal{R} = \cup_{i=0}^{k-1} \mathcal{R}_i$ be a covering subset partition of $\mathcal{R}$ obtained by running Algorithm 5.3.7. Suppose the cell-state vectors are $\boldsymbol{y}_{1,1}^{n'}$ and $\boldsymbol{y}_{2,1}^{n'}$ after the first and second write, respectively. A two-write ICI-free WOM code of length $n'$ is constructed as follows:

1. On the first write, let $m \in [0 : M_1^{\ell} - 1]$ be the information message. Suppose $(m_1, m_2, \ldots, m_{\ell})$ is the $M_1$-ary representation of $m$, i.e., $m = \sum_{i=1}^{\ell} m_i M_1^{\ell-i}$. Then for each $i \in [1 : \ell]$, write $\boldsymbol{y}_{1,(i-1)(n+1)+1}^{i(n+1)-1}$ according to the following rule,

$$\boldsymbol{y}_{1,(i-1)(n+1)+1}^{i(n+1)-1} = f_1(m_i), \forall i \in [1 : \ell];$$

and write $y_{1,i(n+1)}$ according to the following rule

$$y_{1,i(n+1)} = \begin{cases} 1, \text{ if } y_{1,i(n+1)-1} = 1 \text{ and } y_{1,i(n+1)+1} = 1; \\ 0, \text{ otherwise .} \end{cases}$$

2. On the second write, let $m \in [0 : k^{\ell} - 1]$ be the information message. Suppose $(m_1, m_2, \ldots, m_{\ell})$ is the $k$-ary representation of $m$, i.e., $m = \sum_{i=1}^{\ell} m_i k^{\ell-i}$. Then for each $i \in [1 : \ell]$, write $\boldsymbol{y}_{2,(i-1)(n+1)+1}^{i(n+1)-1}$ according to the following rule,

$$\boldsymbol{y}_{2,(i-1)(n+1)+1}^{i(n+1)-1} = \boldsymbol{x}_i \in \mathcal{R}_{m_i},$$

such that $\boldsymbol{x}_i$ covers $\boldsymbol{y}_{1,(i-1)(n+1)+1}^{i(n+1)-1}$;

and write $y_{2,i(n+1)}$ according to the following rule

$$y_{2,i(n+1)} = \begin{cases} 1, \text{ if } y_{2,i(n+1)-1} = 1 \text{ and } y_{2,i(n+1)+1} = 1; \\ 0, \text{ otherwise .} \end{cases}$$

Decoding is simply implemented by reversing the steps of the encoding procedure.

**Remark 5.3.2.** Construction 5.3.2 is a realization of Construction 5.3.1 for $t = 2$. The extension to $t > 2$ is straightforward. Only one "buffer" cell is used to avoid the ICI between adjacent blocks. Note that in Construction 5.3.1, it is possible to decrease the number of "buffer" cells from two to one. Two "buffer" cells are used to simplify the proof in Construction 5.3.1 since the number of "buffer" cells does not affect the asymptotic rate.

Table 5.2 shows the best rate we found using Algorithm 5.3.7 for selected values of $n$. From the table, we see that there exists a sequence of two-write ICI-free WOM codes of rate $R = 1.105 \times \frac{16}{17} \approx 1.04$, which represents 82% of the sum-capacity listed in Table 5.1.

**Table 5.2**: Rates found by Algorithm 5.3.7

| $n$ | 10 | 14 | 16 |
|---|---|---|---|
| $m$ | 2 | 3 | 3 |
| $\lvert\mathcal{L}\rvert$ | 48 | 336 | 513 |
| $k$ | 46 | 139 | 1103 |
| sum-rate | 1.111 | 1.108 | 1.105 |

## 5.4 Conclusion

ICI-free codes are used to mitigate the ICI during programming of flash memories. We extended ICI-free codes in two directions. First, we considered ICI-free balanced codes, which can be used with a dynamic read threshold to adapt to cell-level drift, and determined their asymptotic information rate. We then considered ICI-free WOM codes, which can be used to prolong the flash memory lifetime by reducing the number of block erasures. We calculated the sum-capacity of an ICI-free input-constrained WOM and provided simple code constructions that were used to design several codes with short block lengths. The derivation of the sum-capacity can also be generalized to WOMs with other input constraints.

## Acknowledgments

This chapter is in part a reprint of the material in the paper: Minghai Qin, Eitan Yaakobi, and Paul H. Siegel, "Constrained codes that mitigate intercell interference in read/write cycles for flash memories", IEEE Journal on Selected Areas in Communications, vol. 32, no. 5, pp. 836-846, May 2014.

# Chapter 6

# Parallel programming of flash memories with quantizers

## 6.1 Introduction

Flash memories are a widely-used technology for non-volatile data storage. The basic memory units in flash memories are floating-gate cells, which use charge (i.e., electrons) stored in them to represent data; the amount of charge stored in a cell determines its level. The hot-electron injection mechanism or Fowler-Nordheim tunneling mechanism [12] is used to increase and decrease a cell level by injecting charge into it or by removing charge from it, respectively. The cells in flash memories are organized as blocks, each of which contains about $10^6$ cells. One of the most prominent features of programming flash memory cells is its asymmetry in programming and erasing. That is, increasing a cell level (injecting charge into a cell) is easy to accomplish by applying a certain voltage to the cell. However, decreasing a cell level (removing charge from a cell) is expensive in the sense that the block containing the cell must first be erased, i.e., all charge in the cells within the block is totally removed, before reprogramming them to their target levels. The erase operation, called a block erasure, is not only time-consuming, but also degrades the performance and reduces the longevity of the flash memory [12].

In order to minimize the number of block erasures, programming flash memories is accomplished very carefully using multiple rounds of charge injection to avoid "overshooting" the desired cell level. Therefore, a flash memory can be modeled as a Write Asymmetric Memory (WAM), for which capacity analysis and coding strategies are discussed in [8, 42, 71].

Parallel programming is a crucial tool to increase the write speed when programming

flash memory cells. Two important properties of parallel programming are the use of shared program voltages and the need to account for variation in charge injection [94]. Instead of applying distinct program voltages to different cells, parallel programming applies a common program voltage to many cells simultaneously. Consequently, the write speed is increased and the complexity of hardware realization is substantially reduced. Parallel programming must also account for the fact that cells have different hardness with respect to charge injection [60, 80]. When applying the same program voltage to cells, the amount of charge trapped in different cells may vary. Those cells that have a large amount of trapped charge are called easy-to-program cells and those with little trapped charge are called hard-to-program cells. Understanding this intrinsic property of cells will allow the programming of cells according to their hardness of charge injection. One widely-used programming method is the Incremental Step Pulse Programming (ISPP) scheme [60, 80], which allows easy-to-program cells to be programmed with a lower program voltage and hard-to-program cells to be programmed with a higher program voltage. In this and the following chapters, we study the mathematical models of parallel programmings of flash memories.

In [46, 47], optimized programming for a single flash memory cell was studied. A programming strategy to optimize the precision with respect to two cost functions was proposed, where one of the cost functions is the $\ell_p$ metric and the other is related to rank modulation [48]. It was assumed that the programming noise, which is the difference between the ideal and actual trapped charge in the cell, follows a uniform distribution and the level increment is chosen adaptively according to the current cell level to minimize the cost function.

In [94], algorithms for parallel programming were studied. The underlying model incorporated shared program voltages and variations in cell hardness, as well as a cost function based upon the $\ell_p$ metric. The programming problem was formulated as a special case of the Subspace/Subset Selection Problem (SSP) [35] and the Sparse Approximate Solution Problem (SAS) [66], both of which are NP-hard. Then an algorithm with polynomial time complexity was proposed to search for the optimal programming voltages.

We note that flash memories use multiple discrete levels to represent data in real applications [12]. Hence, if the actual cell level is within a certain distance from the target level, it will be quantized to the correct target level even though there is a gap between them. Read errors can be mitigated by use of error correction codes. If the error correction capability is $e$, then any read error will be totally eliminated as long as the number of read errors is less than $e$. This motivates us to consider another cost function, which is the number of cells that are not quantized correctly to their target levels.

To formulate this more precisely, let $\Theta = (\theta_1, \ldots, \theta_n)$ be the vector of target cell levels and let $\boldsymbol{\ell}_t = (\ell_{1,t}, \ldots, \ell_{n,t})$ be a vector of random variables which represent the level of every cell after $t$ programming rounds. Note that in general the value of $\ell_{i,t}$, for $1 \leqslant i \leqslant n$, depends on the applied voltages, the hardness of the cell, and the programming noise. We will evaluate the performance of any programming method by some cost function $\mathcal{C}(\Theta, \boldsymbol{\ell}_t)$ involving the target cell levels $\Theta$ and the actual cell levels $\boldsymbol{\ell}_t$. Then, the programming problem is to find an algorithm which minimizes the expected value of $\mathcal{C}(\Theta, \boldsymbol{\ell}_t)$. In [94], the cost function was based upon the $\ell_p$ metric, i.e., $\mathcal{C}_p(\Theta, \boldsymbol{\ell}_t) = \left(\sum_{i=1}^{n} \left|\theta_i - \ell_{i,t}\right|^p\right)^{\frac{1}{p}}$. In this chapter, we study a cost function motivated by the quantization of cell levels, namely

$$\mathcal{C}_{\Delta}(\Theta, \boldsymbol{\ell}_t) = \left| \left\{ i \in [n] : \left|\theta_i - \ell_{i,t}\right| > \Delta_i \right\} \right|,$$

where $\Delta_i$ is the quantization distance for the $i$-th cell. We analytically solve the corresponding problem of finding an optimal parallel programming algorithm for the special case where the hardness of each cell is known and there is no programming noise. We also derive optimal programming algorithms for a single cell in the presence of noise, both with and without the availability of feedback during the programming process. We focus upon these scenarios because of their amenability to analysis; the solution to the general cell-programming problem remains open.

Another factor that limits the precision of flash programming is inter-cell interference (ICI), which can cause the level of a cell to increase when its neighboring cells are programmed. The ICI is caused by the parasitic capacitance between neighboring cells, and it is of particular concern in multilevel cell programming [21, 58]. Constrained codes that mitigate the effect of ICI have been studied in [5, 59]. In this chapter, we consider a simple model of ICI and, by application of dynamic programming in the form of the Viterbi algorithm, we derive an optimal, linear-time algorithm for parallel programming with ICI in the absence of noise.

The rest of the chapter is organized as follows. In Section 6.2, we formulate the parallel programming problem with the cost function that reflects the quantization of cell levels. In Section 6.3, we derive a polynomial-time algorithm for optimal parallel programming in the absence of noise, assuming known, deterministic cell-hardness parameters. In Section 6.4, we extend this to a polynomial-time, optimal parallel programming algorithm for the case where ICI is present. In Section 6.5, we study the problem of programming a single cell in the presence of noise, but with no feedback on the cell level during programming. In Section 6.6, we then consider noisy cell programming with applied voltages chosen adaptively using feedback about the current cell level.

## 6.2 Preliminaries

Let us define the cell programming problem in a general information-theoretic framework, such as a cascade channel described in Figure 6.1, where the number of channels is the number of programming rounds, $t$. We assume that there are $n$ cells, denoted by $c_1, c_2, \ldots, c_n$, whose initial levels (i.e., charge levels) are all 0. Each cell is characterized by some target level $\theta_i \geqslant 0$ and the target-level vector is $\Theta \triangleq (\theta_1, \ldots, \theta_n)$. Each round of programming is first described by an encoder $E_i$, $1 \leqslant i \leqslant t$. The input to the first encoder is the target-level vector. For the other encoders, the input also includes feedback on the cell levels after the previous round of programming. The output of encoder $E_i$ is the vector $V_i$ which includes information about the programming voltage of the $i$-th round and the set of cells that are programmed with this voltage. The output of the $i$-th channel, which is the outcome of the $i$-th round of programming, is a function of $V_i$ and $\epsilon_i$ as well as $\boldsymbol{\ell}_{i-1}$ if $i > 1$. The vector $\boldsymbol{\epsilon}_i$ represents the noise in each cell and any other property of the cell that will affect its level. For $i > 1$, the vector $\boldsymbol{\ell}_i$ represents the value of cells after the $i$-th round. For round $i+1$, the outcome of the $i$-th round of programming $\boldsymbol{\ell}_i$ is used to generate a feedback vector $F_i$ on the cell levels to be used in the next round of programming. The goal is to minimize a cost function that measures the difference between the channel output $\boldsymbol{\ell}_t$ after $t$ rounds of programming and the target level vector $\Theta$.

**Remark 6.2.1.** In practice, electrons trapped in the oxide layer can cause transient charge leakage during programming, which leads to a slight decrease of cell levels. Since the leakage and other detrimental factors are typically significantly smaller than the programming noise and inter-cell interference discussed in Section 6.4, we simply assume the cell levels can only increase during programming.

Feedback information on cell levels after a particular write can be used to adaptively choose the programming voltage of the next round, thus increasing the precision of programming. However, obtaining the feedback information is time- and energy-consuming since reading the cell level is accomplished by comparing it to a sequence of reference values.

In the remainder of the chapter, we denote by $[m : n]$ the integer set $\{i \in \mathbb{Z} : m \leqslant i \leqslant n\}$. We will sometimes shorten $[1 : n]$ to $[n]$ when the meaning is clear from the context. We will use $\mathbb{R}_+$ to denote the set of all non-negative real numbers, i.e., $\mathbb{R}_+ = \{x \in \mathbb{R} : x \geqslant 0\}$.

When applying a voltage $V$ to a memory cell $c_i$, where $i \in [n]$, we assume that the increase of the level of cell $c_i$ is linear with $V$, that is, the level of $c_i$ will increase by

$$\alpha V + \epsilon,$$

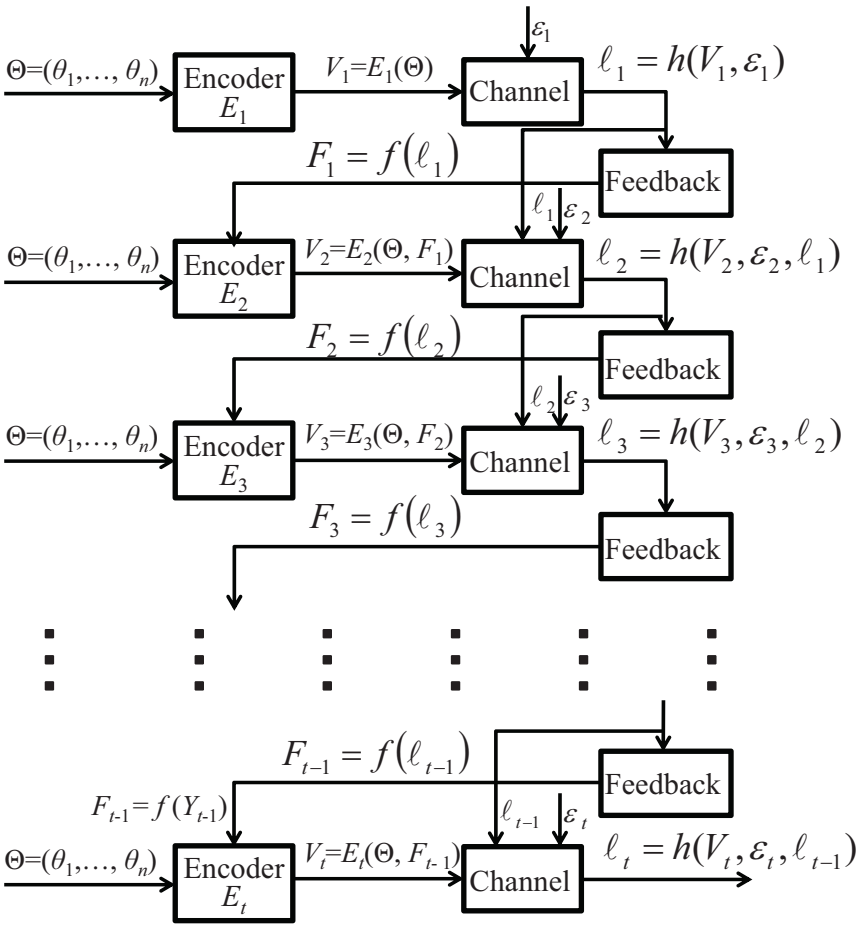


**Figure 6.1**: The information-theoretic framework of the cell programming model.

where $\alpha$ and $\epsilon$ are random variables that might be different for each cell $c_i$ and each round of programming. Each cell $c_i$ is associated with an $\alpha$ and we call it the hardness of charge injection for cell $c_i$, and $\epsilon$ is the programming noise. (Note that the distribution of $\epsilon$ may vary among different cells and different writes.) We define the parallel programming problem in detail as follows.

Let $\Theta = (\theta_1, \ldots, \theta_n)$ be the target cell levels and $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ be the hardness of charge injection and let $\boldsymbol{V} = (V_1, V_2, \ldots, V_t)^T \in \mathbb{R}_+^t$ be the vector of voltages applied on the $t$

rounds of programming. Define the indicator matrix

$$\mathbf{B} = \begin{bmatrix} b_{1,1} & b_{2,1} & \cdots & b_{n,1} \\ b_{1,2} & b_{2,2} & \cdots & b_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ b_{1,t} & b_{2,t} & \cdots & b_{n,t} \end{bmatrix} \in \{0,1\}^{t\times n}$$

where, for $i \in [n]$ and $j \in [t]$, the entry $b_{i,j} \in \{0,1\}$ indicates whether the cell $c_i$ is programmed on the $j$-th round; i.e., $b_{i,j} = 1$ if voltage $V_j$ is applied to cell $c_i$, and $b_{i,j} = 0$, otherwise. Denote the programming noise of the $i$-th cell on the $j$-th programming round by $\epsilon_{i,j}$, for $i \in [n]$ and $j \in [t]$. For $i \in [n]$, let $\ell_{i,t}$ be the random variable representing the level of cell $c_i$ after $t$ rounds of programming; that is,

$$\ell_{i,t} = \sum_{j=1}^{t} \left(\alpha_i V_j + \epsilon_{i,j}\right) b_{i,j}.$$

We define $\boldsymbol{\ell}_t = (\ell_{1,t}, \ldots, \ell_{n,t})$ to be the cell-state vector after $t$ rounds of programming.

We evaluate the performance of the programming by reference to a cost function $\mathcal{C}(\Theta, \boldsymbol{\ell}_t)$. The programming problem is to minimize the expected value of $\mathcal{C}(\Theta, \boldsymbol{\ell}_t)$ over $\boldsymbol{V}$ and $\mathbf{B}$. That is, given $\Theta$, $\boldsymbol{\alpha}$ and $\{\epsilon_{i,j}\}_{n\times t}$, we seek to solve the optimization problem

$$\text{minimize } \mathbb{E}\left[\mathcal{C}(\Theta, \boldsymbol{\ell}_t)\right], \tag{P1}$$

over $\boldsymbol{V} \in \mathbb{R}_+^t$ and $\mathbf{B} \in \{0,1\}^{t\times n}$, where, for a random variable $X$, $\mathbb{E}[X]$ denotes its expected value.

In [94], the $\ell_p$ metric is considered as the cost function, i.e.

$$\mathcal{C}_p(\Theta, \boldsymbol{\ell}_t) = \left(\sum_{i=1}^{n} \left|\theta_i - \ell_{i,t}\right|^p\right)^{\frac{1}{p}},$$

and the optimal solution for (P1) was derived for known $\boldsymbol{\alpha}$ in the absence of noise. However, in real applications, flash memories use multiple discrete levels to represent data and if the cell level $\ell_{i,t}$ is within a certain distance from the target level $\theta_i$, it will be quantized to the correct target level even though there is a gap between $\ell_{i,t}$ and $\theta_i$. This motivates us to consider as the cost function the number of cells that are not correctly quantized to their target levels. To be more precise, letting $\boldsymbol{\Delta} = (\Delta_1, \ldots, \Delta_n)$, we define

$$\mathcal{C}_{\boldsymbol{\Delta}}(\Theta, \boldsymbol{\ell}_t) = \left|\left\{i \in [n] : \left|\theta_i - \ell_{i,t}\right| > \Delta_i\right\}\right|$$

to be the cost function, where $\Delta_i$ is the quantization distance for $c_i$. Therefore, the cell programming problem is to solve

$$\text{minimize } \mathbb{E}\left[\left|\left\{i \in [n] : \left|\theta_i - \ell_{i,t}\right| > \Delta_i\right\}\right|\right], \tag{P2}$$

with $\boldsymbol{V} \in \mathbb{R}_+^t$ and $\mathbf{B} \in \{0,1\}^{t\times n}$.

**Remark 6.2.2.** Problem (P2) is the most general form of the cell programming problem and, therefore, difficult to solve analytically. In the following sections of the chapter, we consider four special cases, of both theoretical and practical interest, for which analytical solutions can be found.

**Remark 6.2.3.** Another concern in programming is the writing speed, which strongly depends on the number of programming rounds. Therefore, an alternative criterion for evaluating the performance of a programming method is the minimum number of programming rounds needed to achieve a specified level of programming accuracy, as described by the expected cost. That is, given the values of $\Theta$, $\boldsymbol{\alpha}$ and $\{\epsilon_{i,j}\}_{n\times t}$, we seek to determine

$$\min_{\boldsymbol{V}\in\mathbb{R}_+^t, \mathbf{B}\in\{0,1\}^{t\times n}} t, \text{ subject to } \mathbb{E}[\mathcal{C}_{\boldsymbol{\Delta}}(\Theta, \boldsymbol{\ell}_t)] \leqslant \gamma, \tag{P2$'$}$$

where $\gamma$ is the maximum allowable expected cost.

If $t$ can be bounded above by a finite number $t_{\max}$, Problem (P2$'$) can be translated to Problem (P2) through a binary search for $t$ between 0 and $t_{\max}$. If there exists an algorithm with time complexity $O(f(n))$ for Problem (P2), then there exists an algorithm with time complexity $O(\log(t_{\max})f(n))$ for (P2$'$). As $t_{\max}$ is usually a small number between 6 and 10 in practice [80, 94], we focus on solving Problem (P2) throughout this chapter.

**Remark 6.2.4.** In practice, quantization is performed by comparing to predetermined voltage levels. The number of such reference levels may therefore affect the read latency, as well as the chip area in a circuit implementation. The trade-off between storage capacity – a function of the number of levels – and quantization speed is determined by the cell quantization distances, $\{\Delta_i\}$. In the case of a uniform quantizer with quantization distance $\Delta$, if the maximum cell level is $\theta_{\max}$, then the number of levels equals $\lfloor \frac{\theta_{\max}}{2\Delta} \rfloor + 1$.

## 6.3 Noiseless Parallel Programming

In this section, we assume that the cell hardness parameters $(\alpha_1, \ldots, \alpha_n)$ are known and deterministic, and there is no programming noise, i.e., $\epsilon_{i,j} = 0, \forall i \in [n], j \in [t]$. In this

scenario, $\ell_{i,t}$ is deterministic so that we can omit the expectation in (P2) and $\ell_{i,t} = \alpha_i \sum_{j=1}^{t} V_j b_{i,j}$. Let $n, t, \boldsymbol{\Delta} = \{\Delta_1, \ldots, \Delta_n\}$ and $\Theta = \{\theta_1, \ldots, \theta_n\}$ denote the block length, the number of programming rounds, the set of quantization distances, and the set of target levels, respectively. Our goal is to find a solution to (P2).

**Lemma 6.3.1.** *The solution to Problem* (P2) *is equivalent to the solution of the following:*

$$\text{maximize } f(\boldsymbol{V}, \mathbf{B}), \tag{P3}$$

*with* $\boldsymbol{V} = (V_1, \ldots, V_t)^T \in \mathbb{R}_+^t$, $\boldsymbol{b}_i = (b_{i,1}, \ldots, b_{i,t})^T \in \{0,1\}^t$ *and* $\mathbf{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$, *where* $u_i = \frac{\theta_i - \Delta_i}{\alpha_i}$, $v_i = \frac{\theta_i + \Delta_i}{\alpha_i}$, $i \in [n]$ *and* $f(\boldsymbol{V}, \mathbf{B}) = \left| \left\{ i \in [n] : u_i \leqslant \boldsymbol{b}_i^T \cdot \boldsymbol{V} \leqslant v_i \right\} \right|$.

**Proof.** The following chain of equations is easily established:

$$\begin{aligned}
&\min_{\boldsymbol{V},\mathbf{B}} \left| \left\{ i \in [n] : \left| \theta_i - \ell_{i,t} \right| > \Delta_i \right\} \right| \\
=& n - \max_{\boldsymbol{V},\mathbf{B}} \left| \left\{ i \in [n] : \left| \theta_i - \ell_{i,t} \right| \leqslant \Delta_i \right\} \right| \\
=& n - \max_{\boldsymbol{V},\mathbf{B}} \left| \left\{ i \in [n] : \left| \frac{\theta_i}{\alpha_i} - \frac{\ell_{i,t}}{\alpha_i} \right| \leqslant \frac{\Delta_i}{\alpha_i} \right\} \right| \\
=& n - \max_{\boldsymbol{V},\mathbf{B}} \left| \left\{ i \in [n] : \left| \frac{\theta_i}{\alpha_i} - \boldsymbol{b}_i^T \cdot \boldsymbol{V} \right| \leqslant \frac{\Delta_i}{\alpha_i} \right\} \right| \\
=& n - \max_{\boldsymbol{V},\mathbf{B}} \left| \left\{ i \in [n] : \frac{\theta_i}{\alpha_i} - \frac{\Delta_i}{\alpha_i} \leqslant \boldsymbol{b}_i^T \cdot \boldsymbol{V} \leqslant \frac{\theta_i}{\alpha_i} + \frac{\Delta_i}{\alpha_i} \right\} \right| \\
=& n - \max_{\boldsymbol{V},\mathbf{B}} \left| \left\{ i \in [n] : u_i \leqslant \boldsymbol{b}_i^T \cdot \boldsymbol{V} \leqslant v_i \right\} \right|,
\end{aligned}$$

where $\boldsymbol{V} \in \mathbb{R}_+^t$, $\mathbf{B} \in \{0,1\}^{t \times n}$, $u_i = \frac{\theta_i - \Delta_i}{\alpha_i}$ and $v_i = \frac{\theta_i + \Delta_i}{\alpha_i}$. This establishes the lemma. ■

The quantities $u_i$ and $v_i$ defined in the statement of Lemma 6.3.1 represent the boundaries of the correct quantization interval for cell $c_i$, $\forall i \in [n]$. We call them the upper threshold point and the lower threshold point for cell $c_i$ and we call the interval $[u_i, v_i]$ the quantization interval for cell $c_i$. Any pair $(\boldsymbol{V}, \mathbf{B})$ that achieves the maximum for (P3) is called an optimal solution pair, and $\boldsymbol{V}$ is called optimal or an optimal solution if there exists $\mathbf{B}$ such that $(\boldsymbol{V}, \mathbf{B})$ is an optimal solution pair.

**Definition 6.3.2.** *Suppose* $u_i$ *and* $v_i$, $i \in [n]$, *are defined as in Lemma* 6.3.1. *Let* $\mathcal{T}_u$ *be the set of upper threshold points and* $\mathcal{T}_v$ *be the set of lower threshold points, i.e.,* $\mathcal{T}_u = \bigcup_{i \in [n]} \{u_i\}$ *and* $\mathcal{T}_v = \bigcup_{i \in [n]} \{v_i\}$. *Let* $\mathcal{T} = \mathcal{T}_u \cup \mathcal{T}_v$ *be the set of all upper and lower threshold points.*

**Example 6.3.1.** Suppose the target levels are $\Theta = (10, 13, 8, 5, 10)$, the quantization distances are $\boldsymbol{\Delta} = (2, 2, 2, 3, 1)$, and the cell hardness parameters are $\boldsymbol{\alpha} = (0.5, 0.5, 1, 1, 0.5)$. According to Lemma 6.3.1, $(u_1, \ldots, u_5) = (16, 22, 6, 2, 18)$ and $(v_1, \ldots, v_5) = (24, 30, 10, 8, 22)$. Then $\mathcal{T} = \{2, 6, 8, 10, 16, 18, 22, 24, 30\}$.

**Remark 6.3.1.** We assume that $|\mathcal{T}| > t$ since otherwise we can easily achieve $\mathcal{C}_{\boldsymbol{\Delta}}(\Theta, \boldsymbol{\ell}_t) = 0$ by using the set of threshold points, $\mathcal{T}$, as the set of programming voltages $\{V_1, \ldots, V_t\}$.

**Definition 6.3.3.** *Suppose $\boldsymbol{V} = (V_1, \ldots, V_t)^T \in \mathbb{R}_+^t$. We define $S_{\boldsymbol{V}}$ to be*

$$S_{\boldsymbol{V}} = \bigcup_{\boldsymbol{b} \in \{0,1\}^t} \{\boldsymbol{b}^T \cdot \boldsymbol{V}\}.$$

*and call it the* **attainable set** *of $\boldsymbol{V}$. That is, $S_{\boldsymbol{V}}$ is the set of voltage values that can be achieved by applying $\boldsymbol{V}$.*

**Remark 6.3.2.** Note that, for any $i \in [n]$, if there exists $z \in S_{\boldsymbol{V}}$ such that $u_i \leqslant z \leqslant v_i$, then there exists $\boldsymbol{b} \in \{0, 1\}^t$ such that $u_i \leqslant \boldsymbol{b}^T \cdot \boldsymbol{V} \leqslant v_i$, and thus the level of $c_i$ can be quantized to the correct target level.

For a fixed $\boldsymbol{V}$, maximizing the function $f(\boldsymbol{V}, \mathbf{B})$ over $\mathbf{B}$ is easy to accomplish by checking whether there exists, for each $i$, an attainable voltage level $z \in S_{\boldsymbol{V}}$ such that $u_i \leqslant z \leqslant v_i$. If one could enumerate all possible vectors $\boldsymbol{V}$, one could use this approach to exhaustively search for an optimal solution. However, since $\boldsymbol{V}$ can be, in principle, any vector in $\mathbb{R}_+^t$, there is an uncountably infinite number of possible choices of $\boldsymbol{V}$ to consider. Nevertheless, Lemma 6.3.4 states that we can limit the number of vectors $\boldsymbol{V}$ under consideration to be polynomial in $n$, and still guarantee that an optimal solution will be found.

**Lemma 6.3.4.** *There exists a matrix $\mathbf{A} \in \{0, 1\}^{t \times t}$, invertible over $\mathbb{R}$, such that*

$$\mathbf{A} \cdot \boldsymbol{V} = \boldsymbol{p},$$

*where $\boldsymbol{p} \in \mathcal{T}^t$ and $\boldsymbol{V}$ is an optimal solution for* (P3).

**Proof.** See Appendix 6.8. ■

**Remark 6.3.3.** In Lemma 6.3.4 and Algorithm 6.3.5 below, the binary matrix $\mathbf{A}$ has to be invertible over $\mathbb{R}$, not necessarily over $GF(2)$. Therefore, enumerating only the binary matrices invertible over $GF(2)$ may not be sufficient to find an optimal solution.

Next we give an algorithm to search for an optimal solution to (P3), which, as we have shown, is also an optimal solution to (P2). First, let $\{p_1, \ldots, p_M\}$ be an arbitrary ordering of the points in $\mathcal{T}$, where $M = |\mathcal{T}|$ is the number of different threshold-point values and $p_i$ can be the value of either an upper or a lower threshold point, for $i \in [M]$. Since $\boldsymbol{p}$ is of length $t$, there are $N = M^t$ choices of $\boldsymbol{p}$ (the entries can be repeated). Let $\{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_N\}$ be an arbitrary ordering of the choices. Now, let $\widetilde{\mathbf{A}} \in \{0,1\}^{t\times t}$ be a binary matrix with distinct rows; the number of such matrices is $Q = \prod_{k=0}^{t-1}(2^t - k)$. Let $\{\widetilde{\mathbf{A}}_1, \ldots, \widetilde{\mathbf{A}}_Q\}$ be an arbitrary ordering of these matrices. Algorithm 6.3.5 will iterate over all choices of $\boldsymbol{p}$ and those matrices $\widetilde{\mathbf{A}}$ that are invertible.

**Algorithm 6.3.5 (Parallel Programming)**

*Function* $(f^*, \boldsymbol{V}^*, \mathbf{B}^*)$ =ParallelProgramming$(t, u_1^n, v_1^n)$.

*Input:*

$t$, $(u_1, \ldots, u_n)$, *and* $(v_1, \ldots, v_n)$.

*Output:*

$f^*$: *maximum value of Problem* (P3);

$(\boldsymbol{V}^*, \mathbf{B}^*)$: *the optimal solution pair.*

*Let* $f^* = 0$;
*Let* $\boldsymbol{V} = \boldsymbol{V}^* = (0, \ldots, 0) \in \mathbb{R}_+^t$,
*Let* $\mathbf{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n) \in \{0,1\}^{t\times n}$, $\boldsymbol{b}_i = \mathbf{0}, \forall i \in [n]$;
*Let* $\mathbf{B}^* \in \{0,1\}^{t\times n}$, $b^*_{i,j} = 0, \forall i \in [t], j \in [n]$;
*For* $i = 1, 2, \ldots, N$ {
    *For* $j = 1, 2, \ldots, Q$ {
        *If* $\widetilde{\mathbf{A}}_j$ *is invertible and* $\widetilde{\mathbf{A}}_j^{-1} \cdot \boldsymbol{p}_i \in \mathbb{R}_+^t$ {
            *Let* $\boldsymbol{V} = \widetilde{\mathbf{A}}_j^{-1} \cdot \boldsymbol{p}_i$;
            *Let* $f = 0$;
            *For* $k = 1, 2, \ldots, n$ {
                *If* $\exists z \in S_{\boldsymbol{V}}$, *such that* $u_k \leqslant z \leqslant v_k$ {
                    *Find* $\boldsymbol{b} \in \{0,1\}^t$, *such that* $\boldsymbol{b}^T \cdot \boldsymbol{V} = z$;
                    *Let* $\boldsymbol{b}_k = \boldsymbol{b}$;
                    $f = f + 1$;
                }
            }
            *If* $f > f^*$
                $f^* = f$, $\boldsymbol{V}^* = \boldsymbol{V}$, $\mathbf{B}^* = \mathbf{B}$;

19. }}}

*Output the optimal solution pair* $(\boldsymbol{V}^*, \mathbf{B}^*)$ *with maximized* $f(\boldsymbol{V}^*, \mathbf{B}^*) = f^*$.

**Example 6.3.2.** Suppose $\boldsymbol{\alpha}, \Theta, \boldsymbol{\Delta}, u_i$ and $v_i, 1 \leqslant i \leqslant 5$, are given as in Example 6.3.1. Suppose the number of programming rounds is $t = 2$. If $\boldsymbol{p}_i = (30, 8)^T \in \mathcal{T}^2$ and

$$\widetilde{\mathbf{A}}_j = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

as we iterate through Line 5 to Line 19, then

$$\boldsymbol{V} = (V_1, V_2)^T = \widetilde{\mathbf{A}}_j^{-1} \cdot \boldsymbol{p}_i = (8, 22)^T,$$

and

$$S_V = \{0, 8, 22, 30\}.$$

By choosing the indicator matrix as

$$\mathbf{B} = \begin{bmatrix} b_{1,1} & b_{2,1} & b_{3,1} & b_{4,1} & b_{5,1} \\ b_{1,2} & b_{2,2} & b_{3,2} & b_{4,2} & b_{5,2} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix},$$

the final cell levels are $\boldsymbol{\ell}_2 = (11, 15, 8, 8, 11)$, where

$$\ell_{i,2} = \alpha_i \sum_{j=1}^{2} V_j b_{i,j} = \alpha_i \left(V_1 b_{i,1} + V_2 b_{i,2}\right), \forall 1 \leqslant i \leqslant 5.$$

It can be easily checked that

$$\theta_i - \Delta_i \leqslant \ell_{i,2} \leqslant \theta_i + \Delta_i, \forall 1 \leqslant i \leqslant 5,$$

implying all cells are correctly quantized and $\boldsymbol{V} = (8, 22)^T$ is an optimal solution.

**Theorem 6.3.6.** *Algorithm* 6.3.5 *finds an optimal solution pair* $(\boldsymbol{V}^*, \mathbf{B}^*)$ *and computes the optimal value* $f(\boldsymbol{V}^*, \mathbf{B}^*)$ *for Problem* (P3). *The time complexity of the algorithm is* $O(n^{t+1})$.

**Proof.** According to Lemma 6.3.4, there exists an optimal solution $(\boldsymbol{V}, \mathbf{B})$, an invertible matrix $\mathbf{A} \in \{0, 1\}^{t \times t}$, and a threshold-point vector $\boldsymbol{p} \in \mathcal{T}^t$, such that

$$\mathbf{A} \cdot \boldsymbol{V} = \boldsymbol{p}.$$

In Algorithm 6.3.5, all possible $\mathbf{A}$'s and $\boldsymbol{p}$'s, have been exhaustively considered and there is at least one optimal $\boldsymbol{V}$ among all the $\boldsymbol{V}$'s derived from $\mathbf{A}$'s and $\boldsymbol{p}$'s. The algorithm outputs the best $\boldsymbol{V}$ among them. This proves that this algorithm will find an optimal solution to (P3).

The number of iterations of the algorithm is of order $NQt^3n2^t$, where $N = M^t \leqslant (2n)^t$ and $Q = \prod_{k=0}^{t-1}(2^t - k)$. Therefore, the complexity is $O(n^{t+1})$. ■

**Remark 6.3.4.** The efficiency of the algorithm could be improved if, rather than iterating over the $Q = \prod_{k=0}^{t-1}(2^t - k)$ binary matrices with distinct rows, we instead iterated only over the set of binary matrices that are invertible over $\mathbb{R}$.

## 6.4 Noiseless Parallel Programming with Inter-cell Interference

In this section, we consider the scenario where cell density has increased to the point that inter-cell interference (ICI) exists. The phenomenon of ICI in flash memories was studied in [21, 58] and constrained codes that mitigate the effect of ICI were presented in [5, 59]. In this section, we extend the results of Sections 6.2 and 6.3, formulating the cell programming problem with ICI as an optimization problem and providing an efficient polynomial time algorithm to solve it.

Suppose the cell layout is a one-dimensional array. When a cell is programmed by applying a voltage to it, the levels of the left and right neighboring cells will also increase. Those cells that cause the ICI are called interfering cells and those cells whose levels are increased unexpectedly because of ICI are called victim cells. Since a large programming voltage will result in a more severe ICI, we make the further assumption that the ICI of the victim cell is proportional to the voltage applied to the interfering cell. We define a sequence of parameters $\beta_{i,i+1} \in \mathbb{R}_+$ and $\beta_{i+1,i} \in \mathbb{R}_+, i \in [n-1]$ to describe the effect of ICI from $c_i$ to $c_{i+1}$ and from $c_{i+1}$ to $c_i$, respectively.

To be more precise, suppose the flash memory cells are $\boldsymbol{c} = (c_1, \ldots, c_n)$ with injection hardness parameters $(\alpha_1, \ldots, \alpha_n)$. There are $t$ rounds of charge injection, corresponding to a set of applied voltages $(V_1, \ldots, V_t)$. There is no programming noise, i.e., $\epsilon_{i,j} = 0, \forall i \in [n], j \in [t]$. If the voltage applied to the cell $c_i$ in round $j$ is $V_j$, then

- the cell level of $c_i$ is increased by $\alpha_i b_{i,j} V_j$, for all $i \in [n]$
- the cell level of $c_{i+1}$ is increased by $\alpha_{i+1} b_{i+1,j} \beta_{i,i+1} V_j$, for all $i \in [n-1]$
- the cell level of $c_{i-1}$ is increased by $\alpha_{i-1} b_{i-1,j} \beta_{i,i-1} V_j$, for all $i \in [2:n]$.

We represent the indicator matrix as $\mathbf{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n) \in \{0,1\}^{t \times n}$, where the vector $\boldsymbol{b}_j$, for $j \in [n]$, denotes the $j$-th column of $\mathbf{B}$. For convenience, we also define $\beta_{0,1} = \beta_{n+1,n} = 0$,

reflecting the fact that the leftmost cell $c_1$ and rightmost $c_n$ have only one neighboring cell each. Similarly, we define $\boldsymbol{b}_0 = (b_{0,1}, \ldots, b_{0,t})^T = \mathbf{0}, \boldsymbol{b}_{n+1} = (b_{n+1,1}, \ldots, b_{n+1,t})^T = \mathbf{0}$.

Now, let $\boldsymbol{\beta}$ denote the vector $(\beta_{0,1}, \beta_{1,2}, \beta_{2,3}, \ldots, \beta_{n,n+1}, \beta_{2,1}, \ldots, \beta_{n,n-1}, \beta_{n+1,n})$. Assuming there is no programming noise, the final cell level of $c_i, i \in [n]$ after $t$ rounds of programming can be written as

$$\ell_{i,t} = \sum_{j=1}^{t} \alpha_i (b_{i,j} + \beta_{i+1,i} b_{i+1,j} + \beta_{i-1,i} b_{i-1,j}) V_j.$$

The cell programming problem is the same as Problem (P2), which is to solve

$$\text{minimize } \mathbb{E}\left[ \left| \left\{ i \in [n] : \left| \theta_i - \ell_{i,t} \right| > \Delta_i \right\} \right| \right],$$

with $\boldsymbol{V} \in \mathbb{R}_+^t$ and $\mathbf{B} \in \{0,1\}^{t \times n}$.

**Lemma 6.4.1.** *The solution to Problem* (P2) *is equivalent to the solution of the following:*

$$\text{maximize } \hat{f}(\boldsymbol{V}, \mathbf{B}), \tag{P3$'$}$$

*with* $\boldsymbol{V} = (V_1, \ldots, V_t)^T \in \mathbb{R}_+^t$, $\boldsymbol{b}_i = (b_{i,1}, \ldots, b_{i,t})^T \in \{0,1\}^t$ *and* $\mathbf{B} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$, *where* $u_i = \frac{\theta_i - \Delta_i}{\alpha_i}$, $v_i = \frac{\theta_i + \Delta_i}{\alpha_i}$, $i \in [n]$ *and*

$$\begin{aligned} &\hat{f}(\boldsymbol{V}, \mathbf{B}) \\ =&\left| \left\{ i \in [n] : u_i \leqslant (\boldsymbol{b}_i^T + \beta_{i+1,i} \boldsymbol{b}_{i+1}^T + \beta_{i-1,i} \boldsymbol{b}_{i-1}^T) \cdot \boldsymbol{V} \leqslant v_i \right\} \right|. \end{aligned}$$

**Proof.** The following chain of equations is easily established.

$$\begin{aligned} &\min_{\boldsymbol{V}, \mathbf{B}} \left| \{ i : |\theta_i - \ell_{i,t}| > \Delta_i, i \in [n] \} \right| \\ =&n - \max_{\boldsymbol{V}, \mathbf{B}} \left| \{ i : |\theta_i - \ell_{i,t}| \leqslant \Delta_i, i \in [n] \} \right| \\ =&n - \max_{\boldsymbol{V}, \mathbf{B}} \left| \left\{ i \in [n] : \left| \frac{\theta_i}{\alpha_i} - \frac{\ell_{i,t}}{\alpha_i} \right| \leqslant \frac{\Delta_i}{\alpha_i} \right\} \right| \\ =&n - \max_{\boldsymbol{V}, \mathbf{B}} \left| \left\{ i \in [n] : \right. \right. \\ &\left. \left. \left| \frac{\theta_i}{\alpha_i} - (\boldsymbol{b}_i^T + \beta_{i+1,i} \boldsymbol{b}_{i+1}^T + \beta_{i-1,i} \boldsymbol{b}_{i-1}^T) \cdot \boldsymbol{V} \right| \leqslant \frac{\Delta_i}{\alpha_i} \right\} \right| \\ =&n - \max_{\boldsymbol{V}, \mathbf{B}} \left| \left\{ i \in [n] : \right. \right. \\ &\left. \left. \frac{\theta_i - \Delta_i}{\alpha_i} \leqslant (\boldsymbol{b}_i^T + \beta_{i+1,i} \boldsymbol{b}_{i+1}^T + \beta_{i-1,i} \boldsymbol{b}_{i-1}^T) \cdot \boldsymbol{V} \leqslant \frac{\theta_i + \Delta_i}{\alpha_i} \right\} \right| \\ =&n - \max_{\boldsymbol{V}, \mathbf{B}} \left| \left\{ i \in [n] : \right. \right. \\ &\left. \left. u_i \leqslant (\boldsymbol{b}_i^T + \beta_{i+1,i} \boldsymbol{b}_{i+1}^T + \beta_{i-1,i} \boldsymbol{b}_{i-1}^T) \cdot \boldsymbol{V} \leqslant v_i \right\} \right|, \end{aligned}$$

where $\boldsymbol{V} \in \mathbb{R}_+^t$, $\mathbf{B} \in \{0,1\}^{t\times n}$, $u_i = \frac{\theta_i - \Delta_i}{\alpha_i}$ and $v_i = \frac{\theta_i + \Delta_i}{\alpha_i}$. ■

As was the case for Problem (P3) in Section 6.3, the optimization in Problem (P3′) is over the applied voltage vector $\boldsymbol{V}$ and the binary indicator matrix $\mathbf{B}$. In Section 6.3, however, there was no ICI and the cells could be treated independently. Consequently, for a fixed voltage vector $\boldsymbol{V}$, we could maximize $f(\boldsymbol{V}, \mathbf{B})$ over $\mathbf{B}$ simply by checking for each $i \in [n]$ individually whether there exists an attainable value $z \in S_{\boldsymbol{V}}$ such that $u_i \leqslant z \leqslant v_i$. The time complexity of this procedure is $O(n)$.

Unfortunately, this procedure is not applicable in the presence of ICI because the level of a given cell depends on the voltage increments applied to neighboring cells. Therefore, to solve Problem (P3′), we first develop an efficient algorithm with time complexity $O(n)$ to maximize $\hat{f}(\boldsymbol{V}, \mathbf{B})$ over $\mathbf{B}$ for a fixed $\boldsymbol{V}$. We then prove a generalization of Lemma 6.3.4 that limits the number of candidate voltage vectors $\boldsymbol{V}$ to be polynomial in $n$, and, finally, present an efficient algorithm to search for the optimal solution pair $(\boldsymbol{V}, \mathbf{B})$.

We call $s_i = (\boldsymbol{b}_{i-1}, \boldsymbol{b}_i) \in \{0,1\}^{t\times 2}$ the state of cell $c_i$, $\forall i \in [n+1]$. Note that the last column of $s_{n+1}$ is all zeros. Let $s_0 \in \{0,1\}^{t\times 2}$ be the all-zero matrix. For example, if $t = 1$, then there are 4 states, corresponding to all binary vectors of length 2, i.e., $(0,0), (0,1), (1,0), (1,1)$. We can relate Problem (P3′) to an optimization problem over a trellis $\mathbf{T}$, which we define as follows [55].

**Definition 6.4.2.** *A trellis* $\mathbf{T}$ *of depth* $n$ *is a triplet* $(\mathbf{S}, \mathbf{E}, \mathbf{L})$, *where* $\mathbf{S} = \mathbf{S}_0 \cup \mathbf{S}_1 \cup \mathbf{S}_2 \cup \cdots \cup \mathbf{S}_n$ *denotes the set of states;* $\mathbf{E} = \mathbf{E}_1 \cup \mathbf{E}_2 \cup \cdots \cup \mathbf{E}_n$ *denotes the set of edges, where each edge* $e \in \mathbf{E}_i$ *has initial state* $\sigma(e) \in \mathbf{S}_{i-1}$ *and terminal state* $\tau(e) \in \mathbf{S}_i$; *and* $\mathbf{L} : \mathbf{E} \to \Sigma_1$ *denotes a label function that assigns to each edge a value in the set* $\Sigma_1$.

For our cell programming problem, we construct a trellis as follows.

**Construction 6.4.1** Suppose the number of cells is $n$ and the number of programming rounds is $t$. We define a trellis $\mathbf{T}$ of depth $n+1$, where $\mathbf{S}_0 = \{\mathbf{0}\}$ and $\mathbf{S}_i$ is the set of states of cell $c_i$, for all $i \in [n+1]$. There exists as edge $e \in \mathbf{E}_i$ from state $s_i \in \mathbf{S}_i$ to state $s_{i+1} \in \mathbf{S}_{i+1}$ if and only if the last column of $s_i$ is the same as the first column of $s_{i+1}$. In that case, $\sigma(e) = s_i$ and $\tau(e) = s_{i+1}$.

We will make use of two label functions. The **terminal state label function** is $\mathbf{L} : \mathbf{E} \to \{0,1\}^t$, where for all $e \in \mathbf{E}$, $\mathbf{L}(e)$ equals the last column of $\tau(e)$. The **branch metric label functions** are $\mathbf{L}_i^b : \mathbf{E}_i \to \{0,1\}, \forall i \in [2 : n+1]$, where $\mathbf{L}_i^b(e) = 1$ if and only if

$e \in \mathbf{E}_i$ and the cell $c_{i-1}$ can be quantized correctly by the voltage vector $\boldsymbol{V}$ and the submatrix $(\sigma(e_i), \mathbf{L}(e_i)) = (\boldsymbol{b}_{i-2}, \boldsymbol{b}_{i-1}, \boldsymbol{b}_i)$ of the indicator matrix, i.e.,

$$u_{i-1} \leqslant (\boldsymbol{b}_{i-1}^T + \beta_{i,i-1}\boldsymbol{b}_i^T + \beta_{i-2,i-1}\boldsymbol{b}_{i-2}^T) \cdot \boldsymbol{V} \leqslant v_{i-1}.$$

Since the construction of the trellis $\mathbf{T}$ depends upon $t, \boldsymbol{V}, \boldsymbol{\beta}, u_1^n$ and $v_1^n$, we denote the trellis by $\mathbf{T}(t, \boldsymbol{V}, \boldsymbol{\beta}, u_1^n, v_1^n)$.

A path $\boldsymbol{e} = (e_1, e_2, \ldots, e_n)$ in $\mathbf{T}$ is a sequence of edges, where $\sigma(e_1) \in \mathbf{S}_0, \sigma(e_{i+1}) = \tau(e_i), \forall i \in [1 : n-1]$. The associated **path metric** is defined as $m(\boldsymbol{e}) = \sum_{i=1}^n \mathbf{L}_i^b(e_i)$. A path $\boldsymbol{e}$ also defines an indicator matrix $\mathbf{B}(\boldsymbol{e})$, which is obtained by reading off and concatenating the column vectors corresponding to the terminal state labels of its edges. The path metric $m(\boldsymbol{e})$ can be interpreted as the number of cells being quantized correctly when the voltage tuple is $\boldsymbol{V}$ and the indicator matrix is $\mathbf{B}(\boldsymbol{e})$, i.e., $m(\boldsymbol{e}) = \hat{f}(\boldsymbol{V}, \mathbf{B}(\boldsymbol{e}))$. Therefore, for a fixed $\boldsymbol{V}$, solving Problem (P3′) is equivalent to finding a path $\boldsymbol{e}$ from $\mathbf{S}_0$ to $\mathbf{S}_n$ that maximizes $m(\boldsymbol{e})$, and thus finding the indicator matrix $\mathbf{B}(\boldsymbol{e})$.

**Example 6.4.1.** Figure 6.2 shows an example of a trellis $\mathbf{T}$ with terminal state label function for $n = 5$ cells and $t = 1$. The number of states for each cell is $2^{2t} = 4$. The number of paths emanating from each state is $2^t = 2$. For the highlighted path $\boldsymbol{e}$, the corresponding indicator matrix $\mathbf{B}(\boldsymbol{e}) = (1, 0, 0, 1, 1)$.

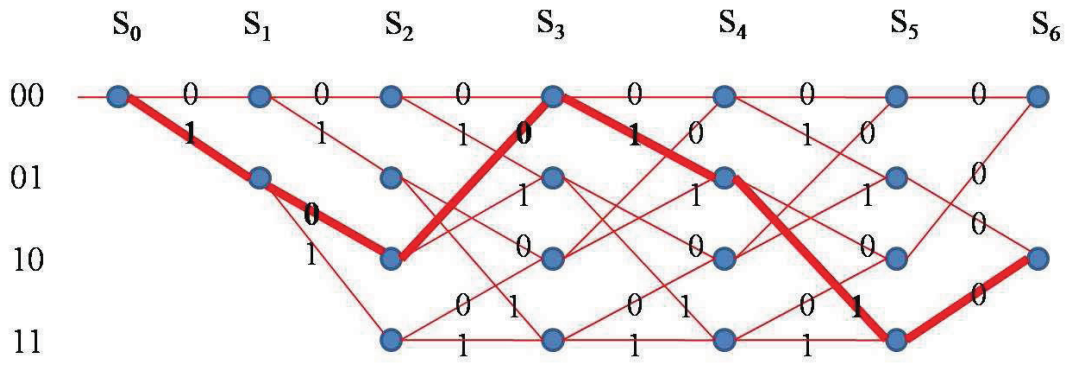


**Figure 6.2**: Illustration of a terminal state label function of a trellis

Next we state the principle of optimality underlying the technique of dynamic programming and, in particular, the well-known Viterbi algorithm.

**Theorem 6.4.3.** *[Principal of Optimality] Let* $\boldsymbol{e} = (e_1, e_2, \ldots, e_{i-1}, e_i)$ *be a path from state* $s_0 \in \mathbf{S}_0$ *to state* $s_i \in \mathbf{S}_i$ *with maximum path metric* $m(\boldsymbol{e})$*. Let* $s_{i-1} \in \mathbf{S}_{i-1}$ *be the terminal state of* $e_{i-1}$*, i.e.,* $s_{i-1} = \tau(e_{i-1})$*. Then, the path* $\widetilde{\boldsymbol{e}} = (e_1, \ldots, e_{i-1})$ *from state* $s_0$ *to* $s_{i-1}$ *has the maximum path metric over all paths from* $s_0$ *to* $s_{i-1}$*.*

Now, we present the Viterbi algorithm as applied to the search for the maximum path metric from $\mathbf{S}_0$ to $\mathbf{S}_n$.

**Algorithm 6.4.4 (Viterbi Algorithm)**

*Function* $(m_i(s), q_i(s)) = \text{Viterbi}(\mathbf{T}(t, \boldsymbol{V}, \boldsymbol{\beta}, u_1^n, v_1^n))$.

*Input:*

*Trellis* $\mathbf{T}(t, \boldsymbol{V}, \boldsymbol{\beta}, u_1^n, v_1^n)$ *in Construction* 6.4.1.

*Output:*

$m_i(s), \forall i \in [n], s \in \mathbf{S}$*: the maximum path metric from* $s_0$ *to* $s \in \mathbf{S}_i$*;*

$q_i(s) \in \mathbf{S}_i, \forall i \in [n], s \in \mathbf{S}$*: the state sequence corresponding to the path from* $s_0$ *to* $s$ *with maximum metric.*

*The algorithm has 4 steps.*

1. *Initialize.*

   *Let* $m_0(s) = 0, \forall s \in \mathbf{S}$*. Let* $q_0(s_0) = s_0$*.*

2. *Add.*

   *For each state* $s \in \mathbf{S}_i$*, and edge* $e \in \mathbf{E}_i$ *such that* $\tau(e) = s$*, let*

   $$\widetilde{m}_i(e) = m_{i-1}(\sigma(e)) + \mathbf{L}_i^b(e)$$

3. *Compare.*

   *For each state* $s \in \mathbf{S}_i$*, determine edge* $e^*$ *with* $\tau(e^*) = s$*, such that* $\widetilde{m}_i(e^*) \geqslant \widetilde{m}_i(e), \forall e$ *such that* $\tau(e) = s$*.*

4. *Select.*

   *Let* $m_i(s) = m_{i-1}(\sigma(e^*)) + \mathbf{L}_i^b(e^*)$ *and* $q_i(s) = (q_{i-1}(\sigma(e^*)), s)$*.*

**Example 6.4.2.** Let $\boldsymbol{\alpha}, \Theta, \boldsymbol{\Delta}, u_i$ and $v_i, 1 \leqslant i \leqslant 5$, be specified as in Example 6.3.1. Suppose $t = 1$, $V_1 = 20$, $\beta_{i+1,i} = \beta_{i+1,i} = 0.2, \forall i \in [n-1]$. Figs. 6.3 and 6.4 show the trellis structure along with the value of the branch metric label function on each edge. Recall that $\boldsymbol{u} = (16, 22, 6, 2, 18)$ and $\boldsymbol{v} = (24, 30, 10, 8, 22)$. The values $m_i(s), s \in \mathbf{S}_i$ are also shown.

The highlighted paths have the maximum path metric from $s_0$ to any state $s_6 \in \mathbf{S}_6$, namely $m_6(10) = 4$, where the indicator matrices are $(1,1,1,0,1)$ and $(1,1,0,0,1)$, respectively.

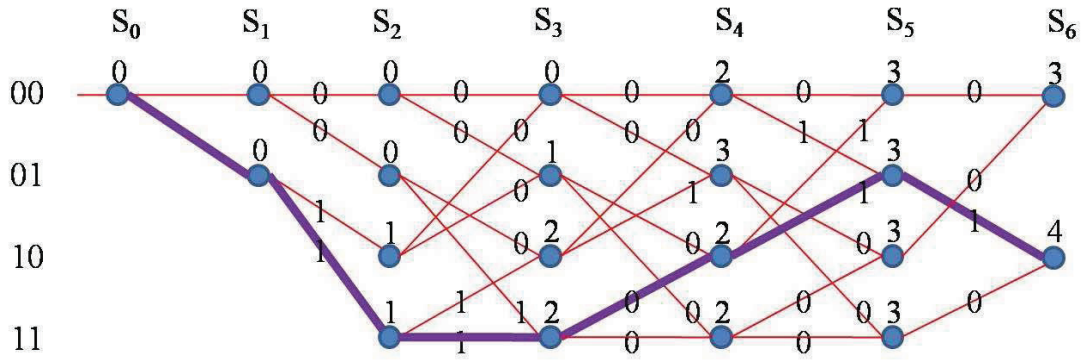


**Figure 6.3**: Path with maximum metric in a trellis with $t = 1$

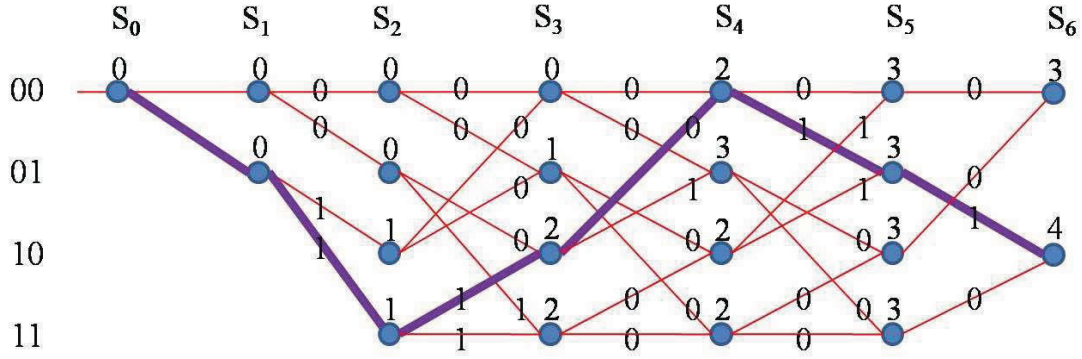


**Figure 6.4**: Another path with maximum metric in a trellis with $t = 1$

**Theorem 6.4.5.** *Algorithm* 6.4.4 *finds the path with the maximum path metric with time complexity* $O(n)$.

**Proof.** The correctness follows from Theorem 6.4.3 and the linear complexity follows from the properties of the Viterbi Algorithm. ■

So far we have constructed an algorithm with linear complexity to determine $\mathbf{B}$ that maximizes $\hat{f}(\boldsymbol{V}, \mathbf{B})$ for a fixed $\boldsymbol{V}$. It is left to determine $\boldsymbol{V}$ that maximizes $\hat{f}(\boldsymbol{V}, \mathbf{B})$.

As in Section 6.3, we define a threshold-point vector $\boldsymbol{p} = (p_{k_1}, \ldots, p_{k_t}) \in \mathcal{T}^t$, where $\mathcal{T}$ is given as in Definition 6.3.2, such that $p_{k_j}$ is a threshold point for the $k_j$-th cell, for $j \in [t]$

and $k_j \in [n]$. For a fixed $\boldsymbol{p}$, we define a finite set of matrices $\mathbf{A}(\boldsymbol{p})$ of size $t \times t$, such that $a_{i,j} \in \{0,\ 1,\ \beta_{k_i+1,k_i},\ 1+\beta_{k_i+1,k_i},\ \beta_{k_i-1,k_i},\ 1+\beta_{k_i-1,k_i},\ \beta_{k_i+1,k_i}+\beta_{k_i-1,k_i},\ 1+\beta_{k_i+1,k_i}+\beta_{k_i-1,k_i}\}$. To determine the optimal $\boldsymbol{V}$, we make use of the following modified version of Lemma 6.3.4.

**Lemma 6.4.6.** *There exists a threshold-point vector $\boldsymbol{p}$ and an invertible matrix $\mathbf{A}(\boldsymbol{p})$ in the corresponding finite set of matrices such that*

$$\mathbf{A}(\boldsymbol{p}) \cdot \boldsymbol{V} = \boldsymbol{p},$$

*where $\boldsymbol{V} \in \mathbb{R}_+^t$ is an optimal solution.*

**Proof.** See Appendix 6.8. ■

**Remark 6.4.1.** Note that, in contrast to Lemma 6.3.4, when ICI is present the matrices that we consider for a given threshold vector $\boldsymbol{p}$ are defined in terms of $\boldsymbol{p}$.

Finally, we give an algorithm to search for an optimal solution to Problem (P3′), which is also an optimal solution to Problem (P2) when ICI exists. Let $\{p_1, \ldots, p_M\}$ be an arbitrary ordering of the points in $\mathcal{T}$, where $M = |\mathcal{T}|$ is the number of different threshold point values. Since $\boldsymbol{p}$ is of length $t$, there are $N = M^t$ choices of $\boldsymbol{p}$ (the entries can be repeated). Let $\{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_N\}$ be an arbitrary ordering of the choices. For a fixed $\boldsymbol{p}_i, i \in [M]$, a sequence of matrices $\widetilde{\mathbf{A}}(\boldsymbol{p}_i)^{t\times t}$ is formed such that no two rows are the same. Thus, the number of different $\widetilde{\mathbf{A}}(\boldsymbol{p}_i)$'s is $Q(\boldsymbol{p}_i) = \prod_{k=0}^{t-1}(8^t - k)$. Let $\{\widetilde{\mathbf{A}}_1(\boldsymbol{p}_i), \ldots, \widetilde{\mathbf{A}}_{Q(\boldsymbol{p}_i)}(\boldsymbol{p}_i)\}$ be an arbitrary ordering of all possible $\widetilde{\mathbf{A}}(\boldsymbol{p}_i)$'s. Algorithm 6.4.7 will iterate over all choices of $\boldsymbol{p}_i$ and those $\widetilde{\mathbf{A}}(\boldsymbol{p}_i)$'s that are invertible.

**Algorithm 6.4.7 (Parallel Programming with ICI)**

*Function* $(f^*, \boldsymbol{V}^*, \mathbf{B}^*) =$ParallelProgrammingICI$(t, u_1^n, v_1^n, \boldsymbol{\beta})$.

*Input:*

$t, (u_1, \ldots, u_n), (v_1, \ldots, v_n)$ *and* $\boldsymbol{\beta}$;

*Output:*

$f^*$: *maximum value of Problem* (P3′);

$(\boldsymbol{V}^*, \mathbf{B}^*)$: *optimal solution pair.*

1. *Let* $f^* = 0$;
2. *Let* $\boldsymbol{V} = \boldsymbol{V}^* = (0, \ldots, 0) \in \mathbb{R}_+^t$;

4. *Let* $\mathbf{B}^* \in \{0,1\}^{t\times n}$, $b^*_{i,j} = 0, \forall i \in [t], j \in [n]$;
5. *For* $i = 1, 2, \ldots, N$ {
6. *For* $j = 1, 2, \ldots, Q(\boldsymbol{p}_i)$ {
7. *If* $\widetilde{\mathbf{A}}_j(\boldsymbol{p}_i)$ *is invertible and* $\widetilde{\mathbf{A}}_j(\boldsymbol{p}_i)^{-1} \cdot \boldsymbol{p}_i \in \mathbb{R}^t_+$ {
8. *Let* $\boldsymbol{V} = \widetilde{\mathbf{A}}_j(\boldsymbol{p}_i)^{-1} \cdot \boldsymbol{p}_i$;
8. *Construct the trellis* $\mathbf{T}(t, \mathbf{V}, \boldsymbol{\beta}, u_1^n, v_1^n)$ *according to Construction* 6.4.1;
9. *Let* $(m_k(s), q_k(s)) = \text{Viterbi}\,(\mathbf{T}(t, \mathbf{V}, \boldsymbol{\beta}, u_1^n, v_1^n))$, *for* $k \in [n], s \in \mathbf{S}$;
10. *Let* $s^* = \arg\max_{s\in\mathbf{S}_n} m_n(s)$ *and* $f = m_n(s^*)$;
11. *If* $f > f^*$ {
12. $f^* = f$, $\boldsymbol{V}^* = \boldsymbol{V}$;
13. *Let the path* $\boldsymbol{e} = (s_0, q_1(s^*), q_2(s^*), \ldots, q_n(s^*))$;
14. *Let* $\mathbf{B}^* = \mathbf{B}(\boldsymbol{e})$;}
15. }}}

*Output the optimal solution pair* $(\boldsymbol{V}^*, \mathbf{B}^*)$ *with maximized* $\hat{f}(\boldsymbol{V}^*, \mathbf{B}^*) = f^*$.

The proof of the following theorem is similar to that of Theorem 6.3.6, so we omit the details.

**Theorem 6.4.8.** *Algorithm* 6.4.7 *finds the optimal solution pair* $(\boldsymbol{V}^*, \mathbf{B}^*)$ *and computes the optimal value* $\hat{f}(\boldsymbol{V}^*, \mathbf{B}^*)$ *for Problem* (P3′). *The time complexity of the algorithm is* $O(n^{t+1})$.

## 6.5 Single Cell Noisy Programming without Feedback

In this section, programming noise is assumed to exist. To carry out our analysis, we must restrict to the case of programming a single cell, with injection hardness $\alpha$. The number of programming rounds is again denoted by $t$, and the programming noise vector $\epsilon_1, \ldots, \epsilon_t$ consists of independently distributed Gaussian random variables with zero means and variances $\sigma_j^2$, $j \in [t]$, respectively.

**Remark 6.5.1.** Note that according to this model, after every programming round the level of each cell could decrease because $\epsilon_j$ could be negative. We choose to study this model while assuming that the variance $\sigma_j$, $j \in [t]$ is much smaller than $\alpha V_j$, i.e., $\mathbf{P}\left(\alpha V_j + \epsilon_j < 0\right)$ is very small. Thus, the probability of decreasing the cell levels is negligible. This model is a reasonable approximation to a physical cell and it can be studied analytically, as will be seen in this section.

Another reasonable assumption we make is that $\sigma_j = \sigma V_j$, $j \in [t]$, where $\sigma$ is a fixed number; that is, the standard deviation of the programming noise is proportional to the programming voltage. This makes sense since larger voltage applied to the cell results in larger power of the programming noise [46]. We further assume that during programming, no feedback information is available, meaning that the actual amount of charge trapped in the cell after each round of programming is not known. The goal is to maximize the probability that after $t$ rounds of programming the final level is in $[\theta - \Delta, \theta + \Delta]$, i.e.,

$$\text{maximize } \mathbf{P}\Big(\theta - \Delta \leqslant \sum_{j=1}^{t} (\alpha V_j + \epsilon_j) \leqslant \theta + \Delta\Big), \tag{P4}$$

with $\boldsymbol{V} \in \mathbb{R}_+^t$.

**Lemma 6.5.1.** *The cell programming problem* (P4) *is equivalent to*

$$\text{maximize } g(\boldsymbol{V}), \tag{P5}$$

*with* $\boldsymbol{V} \in \mathbb{R}_+^t$, *where*

$$g(\boldsymbol{V}) = \frac{1}{\sqrt{2\pi}} \int_{c(\boldsymbol{V})-\delta(\boldsymbol{V})}^{c(\boldsymbol{V})+\delta(\boldsymbol{V})} e^{-u^2/2} \mathrm{d}u,$$

$c(\boldsymbol{V}) = \frac{\theta - \alpha \sum_{j=1}^t V_j}{\sigma \sqrt{\sum_{j=1}^t V_j^2}}$, *and* $\delta(\boldsymbol{V}) = \frac{\Delta}{\sigma \sqrt{\sum_{j=1}^t V_j^2}}$.

**Proof.** We rewrite the probability in (P4) as

$$\begin{aligned} &\mathbf{P}\Big( -\Delta + \theta \leqslant \sum_{j=1}^{t} (\alpha V_j + \epsilon_j) \leqslant \Delta + \theta \Big) \\ =&\mathbf{P}\Big( \frac{-\Delta + \theta - \alpha \sum_{j=1}^t V_j}{\sqrt{\sum_{j=1}^t (\sigma_j^2)}} \leqslant X \leqslant \frac{\Delta + \theta - \alpha \sum_{j=1}^t V_j}{\sqrt{\sum_{j=1}^t (\sigma_j^2)}} \Big), \end{aligned}$$

where $X = \frac{\sum_{j=1}^t (\alpha V_j + \epsilon_j) - \alpha \sum_{j=1}^t V_j}{\sqrt{\sum_{j=1}^t \sigma_j^2}} \sim \mathcal{N}(0,1)$.

Under the assumption that $\sigma_j = \sigma V_j$, we have

$$\begin{aligned} &\mathbf{P}\Big( \frac{-\Delta + \theta - \alpha \sum V_j}{\sigma \sqrt{\sum V_j^2}} \leqslant X \leqslant \frac{\Delta + \theta - \alpha \sum V_j}{\sigma \sqrt{\sum V_j^2}} \Big) \\ &= \frac{1}{\sqrt{2\pi}} \int_{\frac{-\Delta+\theta-\alpha\sum V_j}{\sigma\sqrt{\sum V_j^2}}}^{\frac{\Delta+\theta-\alpha\sum V_j}{\sigma\sqrt{\sum V_j^2}}} e^{-u^2/2} \mathrm{d}u, \\ &= g(\boldsymbol{V}). \end{aligned}$$

■

Let $p(y) = \frac{1}{\sqrt{2\pi}} e^{-y^2/2}$ be the $\mathcal{N}(0,1)$ Gaussian probability density function. Then $g(\boldsymbol{V})$ can be interpreted as the area between the curves $p(y)$ and $y = 0$ on the interval determined by $\boldsymbol{V}$, where the interval is centered at $\frac{\theta - \alpha \sum_{j=1}^{t} V_j}{\sigma \sqrt{\sum_{j=1}^{t} V_j^2}}$, with radius $\frac{\Delta}{\sigma^2 \sqrt{\sum_{j=1}^{t} V_j^2}}$.

**Remark 6.5.2.** In the remainder of the chapter, we will on occasion simplify notation by writing summations of the form $\sum_{j=1}^{t}(\cdot)$ as $\sum(\cdot)$, provided that the meaning is clear from the context.

**Lemma 6.5.2.** *If $\boldsymbol{V}^*$ is the optimal solution to* (P5)*, then $\forall j \in [t], V_j^* = x$, for some constant $x \in R_+$.*

**Proof.** See Appendix 6.9. ■

**Theorem 6.5.3.** *The optimal solution $\boldsymbol{V}^*$ to* (P5) *satisfies the following: $V_j^* = x^*, \forall j \in [t]$, where $x^*$ is the positive root of the equation*

$$\left(2 \ln \frac{b}{a}\right) x^2 + 2(b-a)cx + (a^2 - b^2) = 0,$$

*and $a = \frac{-\Delta+\theta}{\sigma\sqrt{t}}, b = \frac{\Delta+\theta}{\sigma\sqrt{t}}, c = \frac{\alpha\sqrt{t}}{\sigma}$.*

**Proof.**

According to Lemmas 6.5.1 and 6.5.2, the optimal solution to (P5) is achieved by a sequence of programming voltages $\boldsymbol{V}^* = (V_1, \ldots, V_t)$, where $V_j = x, \forall j \in [t]$, for some $x \in R_+$. Referring to the definition of $g(\boldsymbol{V})$, we must therefore find $x \in R_+$ that maximizes

$$h(x) = \int_{\frac{a-cx}{x}}^{\frac{b-cx}{x}} e^{-\frac{u^2}{2}} \mathrm{d}u$$

where $a = \frac{-\Delta+\theta}{\sigma\sqrt{t}}, b = \frac{\Delta+\theta}{\sigma\sqrt{t}}$, and $c = \frac{\alpha\sqrt{t}}{\sigma}$. Note that $h(x) \geqslant 0, \forall x \geqslant 0$. Moreover, $h(0) = 0$ and $h(x) \to 0$ as $x \to 0$. To determine a value of $x$ that maximizes $h(x)$, we examine the points where $h'(x)$, the first derivative of $h(x)$, vanishes. A simple calculation shows that

$$h'(x) = e^{-\frac{1}{2}\left(\frac{b-cx}{x}\right)^2} \cdot \frac{-cx - (b-cx)}{x^2} - e^{-\frac{1}{2}\left(\frac{a-cx}{x}\right)^2} \cdot \frac{-cx - (a-cx)}{x^2}.$$

The condition $h'(x) = 0$ translates to

$$\left(2\ln\frac{b}{a}\right)x^2 + 2(b-a)cx + (a^2 - b^2) = 0.$$

Since $\left(2\ln\frac{b}{a}\right)(a^2 - b^2) < 0$, this equation has two real solutions, one of which is positive. We denote this solution by $x^*$. Noting that $h(x)$ is clearly positive for some $x \in R_+$, we conclude that the maximum value of $h(x)$ must be achieved when $x = x^*$. This completes the proof.

■

**Example 6.5.1.** Using Theorem 6.5.3, a simple calculation shows that the probability of the cell being quantized correctly is a function of three parameters: the number of programming rounds $t$, the ratio between $\theta$ and $\Delta$, and the ratio between $\alpha$ and $\sigma$.

Figure 6.5 shows the probability of correct programming as a function of the number of programming rounds $t$ for different $\sigma$'s, where $\alpha = 1, \theta = 1$, and $\Delta = 0.2$. Figure 6.6 and Figure 6.7 show the minimum number of programming rounds $t$, for different $\theta/\Delta$ and $\alpha/\sigma$, such that the probability of correct quantization is above 90% and 80%, respectively.

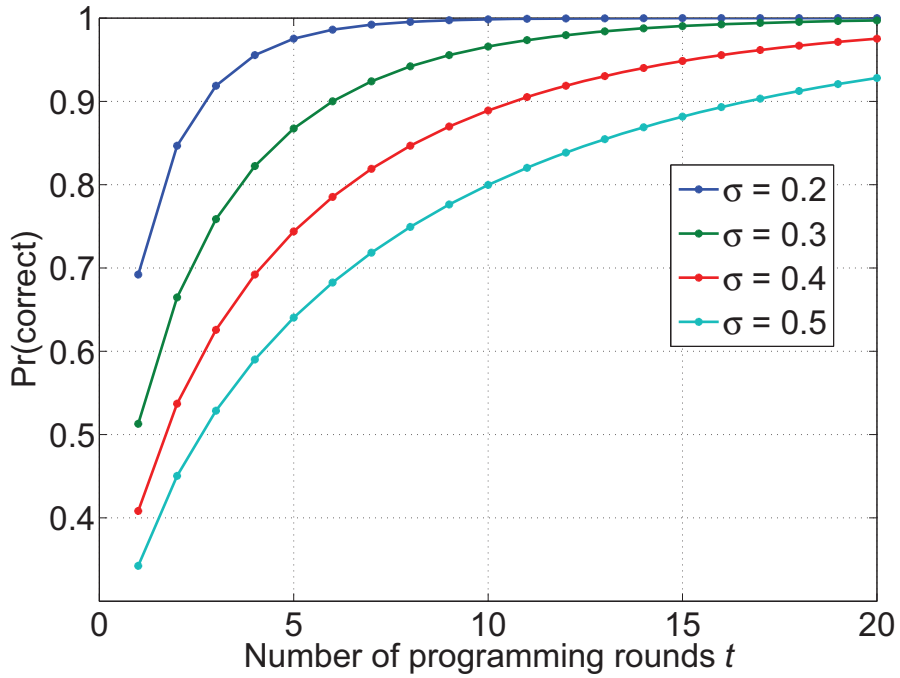


**Figure 6.5**: Probability of correct quantization as a function of the number of programming rounds.

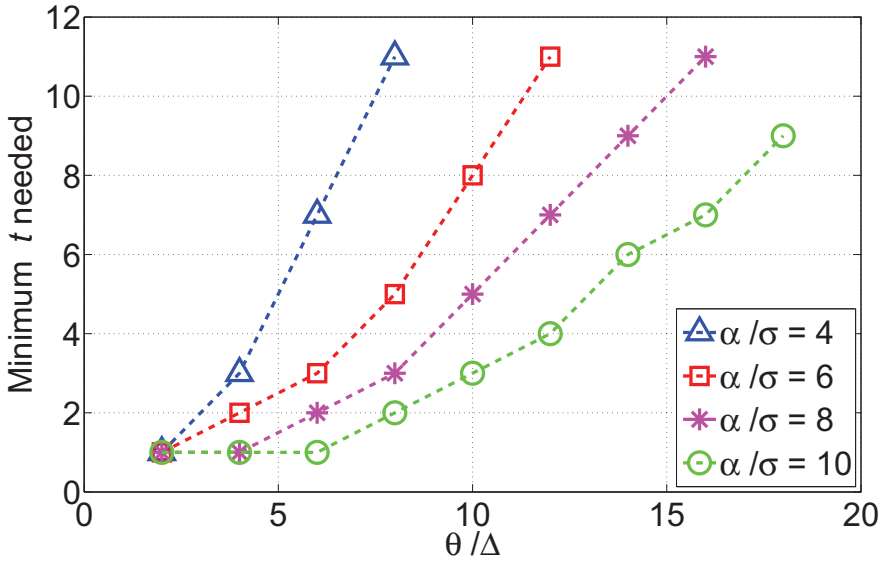


**Figure 6.6**: Minimum number of rounds required to ensure 90% probability of correct programming.

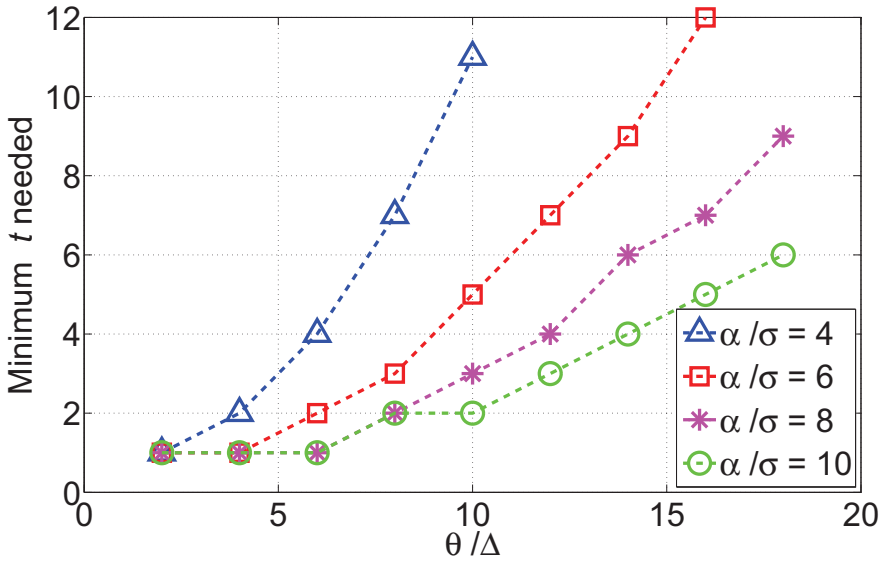


**Figure 6.7**: Minimum number of rounds to ensure 80% probability of correct programming.

## 6.6 Single Cell Noisy Programming with Feedback

In this section, we assume that after every round of programming, we can evaluate the amount of charge that has already been trapped in the cells[1]. That is, we can measure the value $\sum_{j=1}^{k}(\alpha V_j + \epsilon_j)$ after the $k$-th round of programming, $\forall k \in [t]$. Therefore, we can adaptively choose the applied voltages according to the current cell level. Similarly, we assume the injection hardness $\alpha$ of the cell is known and fixed, and the programming noise values $\epsilon_1, \ldots, \epsilon_t$ are independent random variables with probability density functions $p_j(x), \forall j \in [t]$.

Our goal is to maximize the probability that after $t$ rounds of programming the final level is in $[\theta - \Delta, \theta + \Delta]$, i.e.,

$$\text{maximize } \mathbf{P}\Big(\theta - \Delta \leqslant \sum_{j=1}^{t}(\alpha V_j + \epsilon_j) \leqslant \theta + \Delta\Big), \tag{P6}$$

with $\boldsymbol{V} \in \mathbb{R}_+^t$.

**Definition 6.6.1.** *Let $P(\boldsymbol{V}_1^t, \theta, \Delta, t)$ be the probability that the final cell level after $t$ rounds of programming is in $[\theta - \Delta, \theta + \Delta]$ when the voltages are $\boldsymbol{V}_1^t$, where $\boldsymbol{V}_i^j = (V_i, V_{i+1}, \ldots, V_j)$. Let $P(\theta, \Delta, t)$ be the maximum probability over all choices of $\boldsymbol{V}_1^t$, i.e.,*

$$P(\theta, \Delta, t) = \max_{\boldsymbol{V}_1^t \in \mathbb{R}_+^t} P(\boldsymbol{V}_1^t, \theta, \Delta, t),$$

*where*

$$P(\boldsymbol{V}_1^t, \theta, \Delta, t) = \mathbf{P}\left(\theta - \Delta \leqslant \sum_{j=1}^{t}(\alpha V_j + \epsilon_j) \leqslant \theta + \Delta\right).$$

Suppose the target level and quantization distance are $\theta$ and $\Delta$, respectively. Let $P(\theta, \Delta, t)$ be as in Definition 6.6.1. Then we have

$$P(\theta, \Delta, 1) = \max_{V_1 \in \mathbb{R}_+} \int_{\theta-\Delta}^{\theta+\Delta} p_1(x - \alpha V_1)\mathrm{d}x.$$

Suppose $V_1$ is the voltage applied on the first programming round. Then

$$P(\boldsymbol{V}_1^t, \theta, \Delta, t) = \int_{\mathbb{R}_+} p_1(x - \alpha V_1) P(\theta - x, \Delta, t-1)\mathrm{d}x.$$

Since feedback information is available, the recursion

$$P(\theta, \Delta, t) = \max_{V_1 \in \mathbb{R}_+} \int_{\mathbb{R}_+} p_1(x - \alpha V_1) P(\theta - x, \Delta, t-1)\mathrm{d}x$$

[1]Measuring the exact amount of charge injected is time-consuming for real applications, thus it is common to compare the cell level to certain threshold values and to obtain a range for the cell level. In this work, we follow the assumption that the actual cell level is available, as in [46].

holds for $t \geqslant 2$. It follows that the problem of finding $P(\theta, \Delta, t)$ can be reduced to the problem of finding $P(\theta - x, \Delta, t-1)$.

We can compute $P(\theta, \Delta, t)$ numerically using the recursion once we know the distribution of the noise $p_j(x), j \in [t]$. However, analytical results are difficult to derive since the noise distribution $p_j(x), j \in [t]$ could be an arbitrary probability distribution. In the sequel, we assume a simple yet non-trivial noise distribution, namely, $\epsilon_j$ is uniformly distributed over $[\alpha V_j - \delta_1 V_j, \alpha V_j + \delta_2 V_j]$ for $j \in [t]$, where $0 \leqslant \delta_1 \leqslant \alpha$ and $\delta_2 \geqslant 0$. Thus $p_j(x) = \frac{1}{(\delta_1+\delta_2)V_j} I_{x \in [-\delta_1 V_j, \delta_2 V_j]}$. This assumption is similar to the one made in [46] except that we do not constrain $V_j$ to be integer-valued. The size of the support set of the noise distribution is proportional to the programming voltage, which is reasonable since larger voltages result in larger deviations of the noise distribution.

**Lemma 6.6.2.** *In Definition* 6.6.1,

$$P(\theta, \Delta, 1) = \begin{cases} 1, & \text{if } \frac{\theta-\Delta}{\theta+\Delta} < \frac{\alpha-\delta_1}{\alpha+\delta_2} \\ \frac{\alpha+\delta_2}{\delta_1+\delta_2}\frac{2\Delta}{\theta+\Delta} & \text{if } \frac{\theta-\Delta}{\theta+\Delta} \geqslant \frac{\alpha-\delta_1}{\alpha+\delta_2} \end{cases}$$

*and the optimal solution is achieved by* $V_1 = \frac{\theta+\Delta}{\alpha+\delta_2}$.

**Proof.** See Appendix 6.10. ■

Next we would like to find the values of $\boldsymbol{V}_1^t$ that maximize $P(\boldsymbol{V}_1^t, \theta, \Delta, t)$ with feedback information, for arbitrary $t$.

**Lemma 6.6.3.** $P(\theta, \Delta, t)$ *is a non-increasing function of* $\theta$.

**Proof.** See Appendix 6.10. ■

**Theorem 6.6.4.** $P(\boldsymbol{V}_1^t, \theta, \Delta, t)$ *is maximized when* $V_1 = \frac{\theta+\Delta}{\alpha+\delta_2}$.

**Proof.** The proof consists of two parts. First we prove that for any $\widehat{\boldsymbol{V}}_1^t \stackrel{\text{def}}{=} (\widehat{V}_1, \ldots, \widehat{V}_t)$ such that $\widehat{V}_1 < V_1 = \frac{\theta+\Delta}{\alpha+\delta_2}$, $\max_{\widehat{\boldsymbol{V}}_2^t} P(\widehat{\boldsymbol{V}}_1^t, \theta, \Delta, t) \leqslant \max_{\boldsymbol{V}_2^t} P(\boldsymbol{V}_1^t, \theta, \Delta, t)$. Next, we prove that for any $\widetilde{\boldsymbol{V}}_1^t \stackrel{\text{def}}{=} (\widetilde{V}_1, \ldots, \widetilde{V}_t)$ such that $\widetilde{V}_1 > V_1 = \frac{\theta+\Delta}{\alpha+\delta_2}$, $\max_{\widetilde{\boldsymbol{V}}_2^t} P(\widetilde{\boldsymbol{V}}_1^t, \theta, \Delta, t) \leqslant \max_{\boldsymbol{V}_2^t} P(\boldsymbol{V}_1^t, \theta, \Delta, t)$.

Case (1): Suppose $\widehat{V}_1 < V_1 = \frac{\theta+\Delta}{\alpha+\delta_2}$.

First we provide a sketch of the proof. If the first voltage applied is $V_1$ (resp. $\widehat{V}_1$), then the cell level after the first programming round is uniformly distributed over $\mathbb{F} = [(\alpha -$

$\delta_1)V_1, (\alpha+\delta_2)V_1]$ (resp. $\widehat{\mathbb{F}} = [(\alpha-\delta_1)\widehat{V}_1, (\alpha+\delta_2)\widehat{V}_1]$). We will divide $\mathbb{F}$ (resp. $\widehat{\mathbb{F}}$) into non-overlapping intervals and prove that in each interval applying $V_1$ yields higher probability of correct programming than applying $\widehat{V}_1$.

Let $\ell = \lceil \frac{(\delta_1+\delta_2)\widehat{V}_1}{(\alpha-\delta_1)(V_1-\widehat{V}_1)} \rceil$ and divide $\mathbb{F}$ (resp. $\widehat{\mathbb{F}}$) evenly into $\ell$ non-overlapping intervals. That is, let $\mathbb{F}_i = \big[(\alpha-\delta_1)V_1 + (i-1)\frac{(\delta_1+\delta_2)V_1}{\ell}, (\alpha-\delta_1)V_1 + i\frac{(\delta_1+\delta_2)V_1}{\ell}\big]$ (resp. $\widehat{\mathbb{F}}_i = \big[(\alpha-\delta_1)\widehat{V}_1 + (i-1)\frac{(\delta_1+\delta_2)\widehat{V}_1}{\ell}, (\alpha-\delta_1)\widehat{V}_1 + i\frac{(\delta_1+\delta_2)\widehat{V}_1}{\ell}\big]$), for $i \in [\ell]$. Note that if $x \in \mathbb{F}_i$ and $\hat{x} \in \widehat{\mathbb{F}}_i$, then $x > \hat{x}, \forall i \in [\ell]$. Then

$$\begin{aligned}
&\max_{\boldsymbol{V}_2^t} P(\boldsymbol{V}_1^t, \theta, \Delta, t) = \int_{\mathbb{F}} p_1(x-\alpha V_1) P(\theta - x, \Delta, t-1)\mathrm{d}x \\
&= \int_{\bigcup_{i=1}^{\ell}\mathbb{F}_i} \frac{1}{(\delta_1+\delta_2)V_1} P(\theta - x, \Delta, t-1)\mathrm{d}x \\
&= \sum_{i=1}^{\ell} \int_{\mathbb{F}_i} \frac{1}{(\delta_1+\delta_2)V_1} P(\theta - x, \Delta, t-1)\mathrm{d}x
\end{aligned}$$

and

$$\begin{aligned}
&\max_{\widehat{\boldsymbol{V}}_2^t} P(\widehat{\boldsymbol{V}}_1^t, \theta, \Delta, t) = \int_{\mathbb{F}} p_1(x-\alpha \widehat{V}_1) P(\theta - x, \Delta, t-1)\mathrm{d}x \\
&= \int_{\bigcup_{i=1}^{\ell}\widehat{\mathbb{F}}_i} \frac{1}{(\delta_1+\delta_2)\widehat{V}_1} P(\theta - x, \Delta, t-1)\mathrm{d}x \\
&= \sum_{i=1}^{\ell} \int_{\widehat{\mathbb{F}}_i} \frac{1}{(\delta_1+\delta_2)\widehat{V}_1} P(\theta - x, \Delta, t-1)\mathrm{d}x.
\end{aligned}$$

According to Lemma 6.6.3, $P(\theta - x, \Delta, t-1)$ is a non-decreasing function of $x$; therefore, for each element in the summation, we have

$$\begin{aligned}
&\int_{\mathbb{F}_i} \frac{1}{(\delta_1+\delta_2)V_1} P(\theta - x, \Delta, t-1)\mathrm{d}x \\
&\geqslant \frac{|\mathbb{F}_i| P(\theta - ((\alpha-\delta_1)V_1 + i\frac{(\delta_1+\delta_2)V_1}{\ell}), \Delta, t-1)}{(\delta_1+\delta_2)V_1} \\
&\geqslant \frac{|\widehat{\mathbb{F}}_i| P(\theta - ((\alpha-\delta_1)\widehat{V}_1 + (i-1)\frac{(\delta_1+\delta_2)\widehat{V}_1}{\ell}), \Delta, t-1)}{(\delta_1+\delta_2)\widehat{V}_1} \\
&\geqslant \int_{\widehat{\mathbb{F}}_i} \frac{1}{(\delta_1+\delta_2)\widehat{V}_1} P(\theta - x, \Delta, t-1)\mathrm{d}x,
\end{aligned}$$

for all $i \in [\ell]$. This proves $\max_{\widehat{\boldsymbol{V}}_2^t} P(\widehat{\boldsymbol{V}}_1^t, \theta, \Delta, t) \leqslant \max_{\boldsymbol{V}_2^t} P(\boldsymbol{V}_1^t, \theta, \Delta, t)$.

Case (2): Suppose $\widetilde{V}_1 > V_1 = \frac{\theta+\Delta}{\alpha+\delta_2}$.

If the first voltage applied is $\widetilde{V}_1$, then the voltage of the cell after the first round of programming is uniformly distributed over $\widetilde{\mathbb{F}} = [(\alpha-\delta_1)\widetilde{V}_1, (\alpha+\delta_2)\widetilde{V}_1]$. Once the voltage is in $[\theta+\Delta, (\alpha+\delta_2)\widetilde{V}_1]$, the probability that after $t$ rounds of programming the final cell level is within the interval $[\theta-\Delta, \theta+\Delta]$ is 0, since the cell level cannot be decreased in our model of flash cell programming.

Now, since $\widetilde{V}_1 > V_1$, we have

$$\begin{aligned}
&\max_{\widetilde{\boldsymbol{V}}_2^t} P(\widetilde{\boldsymbol{V}}_1^t, \theta, \Delta, t) = \int_{\widetilde{\mathbb{F}}} p_1(x-\alpha\widetilde{V}_1) P(\theta-x, \Delta, t-1)\mathrm{d}x \\
&= \int_{(\alpha-\delta_1)\widetilde{V}_1}^{\theta+\Delta} \frac{1}{(\delta_1+\delta_2)\widetilde{V}_1} P(\theta-x, \Delta, t-1)\mathrm{d}x \\
&\leqslant \int_{(\alpha-\delta_1)\widetilde{V}_1}^{\theta+\Delta} \frac{1}{(\delta_1+\delta_2)V_1} P(\theta-x, \Delta, t-1)\mathrm{d}x \\
&\leqslant \int_{(\alpha-\delta_1)V_1}^{\theta+\Delta} \frac{1}{(\delta_1+\delta_2)V_1} P(\theta-x, \Delta, t-1)\mathrm{d}x \\
&= \max_{\boldsymbol{V}_2^t} P(\boldsymbol{V}_1^t, \theta, \Delta, t).
\end{aligned}$$

Noting that

$$\max_{\boldsymbol{V}_1^t} P(\boldsymbol{V}_1^t, \theta, \Delta, t) = \max_{V_1} \max_{\boldsymbol{V}_2^t} P(\boldsymbol{V}_1^t, \theta, \Delta, t),$$

we conclude that $P(\boldsymbol{V}_1^t, \theta, \Delta, t)$ is maximized when $V_1 = \frac{\theta+\Delta}{\alpha+\delta_2}$. ■

Next we give an algorithm for determining the optimal cell programming for Problem (P6), where feedback information is available.

**Algorithm 6.6.5** *The voltage $V_j$ on the $j$-th round of programming, where $1 \leqslant j \leqslant t$, is set as follows.*

*Let $x_j$ denote the feedback representing the cell level before the $j$-th write, where, for $j = 1$, we set $x_1 = 0$.*

*Set $V_j = \frac{\theta - x_j + \Delta}{\alpha+\delta_2}$.*

**Corollary 6.6.6.** *Algorithm* 6.6.5 *gives an optimal solution for the cell programming problem* (P6).

**Proof.** According to Theorem 6.6.4, if we need to reach the level $\theta$, then the voltage applied on the first round is $\frac{\theta+\Delta}{\alpha+\delta_2}$. Thus, after the $(j-1)$-st round, if we know that the current cell level is $x_j$, then the voltage applied on the $j$-th round is $\frac{\theta-x_j+\Delta}{\alpha+\delta_2}$, which completes the proof. ■

## 6.7 Conclusion

Accurate and efficient cell programming is critical to the enhancement of flash memory functionality and storage capacity. Programming techniques must take into account the asymmetric nature of the write process, the manner in which discrete data values are represented within the range of cell levels, the presence or absence of noise, and the reduction in write latency that parallel programming can provide.

In this chapter, we make the realistic assumption that cell levels are quantized to a discrete set of levels to represent digital data. The programming of a cell is considered to be successful if the programmed cell level is correctly quantized to the desired target level. For several scenarios, we present programming algorithms that, for a specified number of programming rounds, achieve optimality with respect to this figure of merit. Specifically, when cells have known hardness to charge injection and the programming process is noiseless, we derive an optimal parallel programming algorithm whose complexity is polynomial in the number of cells. We also modify the algorithm to take into account the presence of inter-cell interference from adjacent cells.

We also consider techniques for programming a single cell in the presence of noise. Assuming that no feedback on the cell level is available during the write process, we present a programming algorithm that, for a given number of programming rounds, maximizes the probability of attaining a cell level corresponding to the desired target level. We then address the situation where feedback is available and present an optimal strategy for adaptively choosing the programming voltages.

## 6.8 Appendix A

**Proof of Lemma 6.3.4.** We prove the lemma by induction.

For $t = 1$, it is equivalent to prove that there exists an optimal $V_1$ such that $V_1 \in \mathcal{T}$. So, suppose $V_1^*$ is an optimal solution. If $V_1^* \in \mathcal{T}$, then the lemma holds for $t = 1$; if not, define

$\delta_{min}$ to be the smallest distance from $V_1^*$ to an element of $\mathcal{T}$, i.e.,

$$\delta_{min} = \min_{p \in \mathcal{T}} \{|V_1^* - p|\}.$$

If $\delta_{min}$ is achieved by choosing an upper threshold point, set $\widehat{V}_1 = V_1^* + \delta_{min}$; otherwise, set $\widehat{V}_1 = V_1^* - \delta_{min}$. That is, $\widehat{V}_1$ is the closest threshold point to $V_1^*$. By the definition of $\delta_{min}$, any cell that can be quantized correctly using $V_1^*$ can be quantized correctly using $\widehat{V}_1$; thus, $\widehat{V}_1$ is also an optimal solution. Meanwhile, $\widehat{V}_1 \in \mathcal{T}$. This proves that there always exists an optimal solution $V_1 \in \mathcal{T}$.

Suppose the lemma holds if the number of programming rounds is $t-1$. That is, for $t \geqslant 2$, assume there exists an invertible matrix $\mathbf{A} \in \{0,1\}^{(t-1)\times(t-1)}$, such that

$$\mathbf{A}\boldsymbol{V} = \boldsymbol{p},$$

where $\boldsymbol{V} \in \mathbb{R}_+^{t-1}$ is an optimal solution for (P3) and $\boldsymbol{p} \in \mathcal{T}^{(t-1)}$. We are going to prove by contradiction that the lemma holds if the number of programming rounds is $t$.

Suppose the opposite is true. Then, for any $\boldsymbol{p}' \in \mathcal{T}^t$, there does not exist an invertible matrix $\mathbf{A} \in \{0,1\}^{t\times t}$, such that

$$\mathbf{A}\boldsymbol{V} = \boldsymbol{p},$$

where $\boldsymbol{V}$ is an optimal solution for (P3). Let $t'$ be the largest number, $0 \leqslant t' < t$, such that there exists a matrix $\mathbf{A}' \in \{0,1\}^{t'\times t}$ with full row rank, such that

$$\mathbf{A}'\boldsymbol{V}^* = \boldsymbol{p}',$$

where $\boldsymbol{V}^*$ is an optimal solution for (P3) and $\boldsymbol{p}' \in \mathcal{T}^{t'}$.

Let $\boldsymbol{V} \in \mathbb{R}_+^t$ satisfy $\mathbf{A}'\boldsymbol{V} = \boldsymbol{p}'$. Since $\text{rank}(\mathbf{A}') = t' < t$, the solution space for $\boldsymbol{V}$ is a non-empty polytope $\mathcal{P}$ consisting of the non-negative vectors in a $t'$-dimensional subspace. That is,

$$\mathcal{P} = \left\{\boldsymbol{V} \in \mathbb{R}_+^t | \mathbf{A}'\boldsymbol{V} = \boldsymbol{p}'\right\}.$$

(Note that if $t' = 0$, then $\mathcal{P}$ is the space of non-negative $t$-dimensional vectors.)

**Claim 6.8.1.** *There exists a $\widehat{\boldsymbol{V}}$ in $\mathcal{P}$ such that $\widehat{\boldsymbol{V}}$ is on the boundary of $\mathcal{P}$, i.e., $\exists \widehat{\boldsymbol{V}} \in \mathcal{P}$ and $k \in [t]$, such that $\widehat{V}_k = 0$.*

**Proof.** Since $\mathcal{P}$ is a non-trivial polytope, there exists $\boldsymbol{X} \in \mathbb{R}_+^t$ in $\mathcal{P}$ such that $\boldsymbol{X} \neq \boldsymbol{V}^*$. If there exists $j \in [t]$ such that $X_j < V_j^*$, let

$$z_{\min} = \arg\min_{1 \leqslant j \leqslant t} \frac{X_j}{V_j^*}.$$

If there exists more than one index $j \in [t]$ that minimizes $\frac{X_j}{V_j^*}$, then $z_{\min}$ is chosen arbitrarily from among these indices. Let

$$y_{\min} = \min_{1 \leqslant j \leqslant t} \frac{X_j}{V_j^*} = \frac{X_{z_{\min}}}{V_{z_{\min}}^*} < 1,$$

and set

$$\widehat{\boldsymbol{V}} = \frac{\boldsymbol{X} - y_{\min} \boldsymbol{V}^*}{(1 - y_{\min})}.$$

Since $\widehat{\boldsymbol{V}}$ is a linear combination of $\boldsymbol{X}$ and $\boldsymbol{V}^*$, we have $\mathbf{A}'\widehat{\boldsymbol{V}} = \boldsymbol{p}'$. Then

$$\widehat{V}_{z_{\min}} = \frac{X_{z_{\min}} - y_{\min} V_{z_{\min}}^*}{(1 - y_{\min})} = \frac{X_{z_{\min}} - \frac{X_{z_{\min}}}{V_{z_{\min}}^*} V_{z_{\min}}^*}{(1 - y_{\min})} = 0,$$

and

$$\widehat{V}_j = \frac{X_j - y_{\min} V_j^*}{(1 - y_{\min})} \geqslant 0, \forall j \in [t].$$

Therefore $\widehat{\boldsymbol{V}} \in \mathcal{P}$ and $\widehat{V}_{z_{\min}} = 0$.

If there does not exist $j \in [t]$ such that $X_j < V_j^*$, then there exists $j \in [t]$ such that $V_j^* < X_j$ since $\boldsymbol{X} \neq \boldsymbol{V}^*$. Following similar reasoning, we can prove that there exists $\widehat{\boldsymbol{V}} \in \mathcal{P}$ such that $\widehat{V}_k = 0$, for some $k \in [t]$. ■

Now, there are two different cases to consider for the $\widehat{\boldsymbol{V}}$ of Claim 6.8.1.

**Case (1): $\widehat{\boldsymbol{V}}$ is an optimal solution of (P3).**

**Claim 6.8.2.** *If $\widehat{\boldsymbol{V}}$ is optimal, then all of the $n$ cells can be quantized correctly using $t-1$ rounds of programming.*

**Proof.** Suppose the opposite is true, and there exists $c_i$ with quantization interval $[v_i, u_i]$ that is not quantized correctly by $\widehat{\boldsymbol{V}}$. Set $\widetilde{V}_j = \widehat{V}_j$ for all $j$ such that $1 \leqslant j \neq k \leqslant t$ and set $\widetilde{V}_k = v_i$. Then the number of cells that are quantized correctly by $\widetilde{\boldsymbol{V}}$ is larger than $\widehat{\boldsymbol{V}}$, contradicting the assumption that $\widehat{\boldsymbol{V}}$ is optimal. ■

By the induction assumption, if the number of programming rounds is $t-1$, there exists an invertible matrix $\mathbf{A} \in \{0,1\}^{(t-1)\times(t-1)}$, such that

$$\mathbf{A}\boldsymbol{V} = \boldsymbol{p},$$

where $\boldsymbol{V} \in \mathbb{R}_+^{(t-1)}$ is an optimal solution and $\boldsymbol{p} \in \mathcal{T}^{(t-1)}$. Note that in this case, according to Claim 6.8.2, all of the $n$ cells can be quantized correctly. We form another invertible matrix

$\widetilde{\mathbf{A}} \in \{0,1\}^{t\times t}$ by adding one column and one row to $\mathbf{A}$, where all added entries are 0 except that $\widetilde{a}_{t,t} = 1$. Let $\widetilde{\boldsymbol{V}} = (V_1, \ldots, V_{t-1}, p_t)^T \in \mathbb{R}_+^t$ where $p_t \in \mathcal{T}$ is any threshold point. Let $\widetilde{\boldsymbol{p}} = (\boldsymbol{p}^T, p_t)^T = (p_1, \ldots, p_{t-1}, p_t)^T$ be a threshold-point vector. Then we have

$$\widetilde{\mathbf{A}}\widetilde{\boldsymbol{V}} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0^T} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{V} \\ p_t \end{bmatrix} = \begin{bmatrix} \boldsymbol{p} \\ p_t \end{bmatrix} \in \mathcal{T}^t.$$

and $\widetilde{\boldsymbol{V}}$ is optimal since all of the $n$ cells are quantized correctly. The existence of $\widetilde{\mathbf{A}}$, $\widetilde{\boldsymbol{V}}$ and $\widetilde{\boldsymbol{p}}$ contradicts the assumption that $t'$ is the maximum row rank.

**Case (2): $\widehat{\boldsymbol{V}}$ is not an optimal solution of (P3).**

Note that because $\boldsymbol{V}^*$ is optimal while $\widehat{\boldsymbol{V}}$ is not optimal, the following claims hold.

**Claim 6.8.3.** *There exists $\boldsymbol{b} \in \{0,1\}^t$ such that $\boldsymbol{b}^T\boldsymbol{V}^* \in [v_i, u_i]$ and $\boldsymbol{b}^T\widehat{\boldsymbol{V}} \notin [v_i, u_i]$, for some $i \in [n]$.*

**Proof.** Since $\boldsymbol{V}^*$ is optimal while $\widehat{\boldsymbol{V}}$ is not optimal, at least one cell can be quantized correctly by $\boldsymbol{V}^*$ but not by $\widehat{\boldsymbol{V}}$. Suppose the cell is $c_i$. Then there exists $\boldsymbol{b} \in \{0,1\}^t$ such that $\boldsymbol{b}^T\boldsymbol{V}^* \in [v_i, u_i]$ and $\boldsymbol{b}^T\widehat{\boldsymbol{V}} \notin [v_i, u_i]$. ■

**Claim 6.8.4.** *Every $\boldsymbol{b} \in \{0,1\}^t$ satisfying the property in Claim* 6.8.3 *is linearly independent with respect to the set of row vectors of $\mathbf{A}'$.*

**Proof.** Suppose the opposite is true. That is, $\boldsymbol{b}^T = \boldsymbol{x}\mathbf{A}'$, for some $\boldsymbol{x} \in \mathbb{R}^{t'}$. Then

$$\boldsymbol{b}^T\boldsymbol{V}^* = \boldsymbol{x}\mathbf{A}'\boldsymbol{V}^* = \boldsymbol{x}\boldsymbol{p}' = \boldsymbol{x}\mathbf{A}'\widehat{\boldsymbol{V}} = \boldsymbol{b}^T\widehat{\boldsymbol{V}}.$$

This contradicts the fact that $\boldsymbol{b}^T\boldsymbol{V}^* \neq \boldsymbol{b}^T\widehat{\boldsymbol{V}}$. ■

Suppose the number of triplets $(\boldsymbol{b}, v_i, u_i)$ for $\boldsymbol{b} \in \{0,1\}^t$ and $i \in [n]$ in Claim 6.8.3 is $K$. We list all such triplets and label them by $(\boldsymbol{b}_k, w_k, y_k)$, $k \in [K]$. Note that one and only one of $w_k$ and $y_k$ is between $\boldsymbol{b}_k^T\boldsymbol{V}^*$ and $\boldsymbol{b}_k^T\widehat{\boldsymbol{V}}$. Without loss of generality, we assume that $w_k$ is between $\boldsymbol{b}_k^T\boldsymbol{V}^*$ and $\boldsymbol{b}_k^T\widehat{\boldsymbol{V}}$. In particular, $\boldsymbol{b}_k^T\widehat{\boldsymbol{V}} \leqslant w_k \leqslant \boldsymbol{b}_k^T\boldsymbol{V}^*$. Define

$$\delta_{\min} = \min_{1\leqslant k\leqslant K} \frac{\boldsymbol{b}_k^T\boldsymbol{V}^* - w_k}{\boldsymbol{b}_k^T\boldsymbol{V}^* - \boldsymbol{b}_k^T\widehat{\boldsymbol{V}}} \in [0,1]$$

and

$$k_{\min} = \arg \min_{1 \leqslant k \leqslant K} \frac{\boldsymbol{b}_k^T \boldsymbol{V}^* - w_k}{\boldsymbol{b}_k^T \boldsymbol{V}^* - \boldsymbol{b}_k^T \widehat{\boldsymbol{V}}}.$$

Consider the convex combination of $\boldsymbol{V}^*$ and $\widehat{\boldsymbol{V}}$ given by

$$\widetilde{\boldsymbol{V}} = \boldsymbol{V}^* - \delta_{\min}(\boldsymbol{V}^* - \widehat{\boldsymbol{V}}) = (1 - \delta_{\min})\boldsymbol{V}^* + \delta_{\min}\widehat{\boldsymbol{V}}.$$

**Claim 6.8.5.** *$\widetilde{\boldsymbol{V}}$ is an optimal solution of (*P3*).*

**Proof.** Suppose $c_i$ can be programmed into its quantization interval $[v_i, u_i], i \in [n]$ by $\boldsymbol{V}^*$ but not by $\widehat{\boldsymbol{V}}$. According to Claim 6.8.3, let $\boldsymbol{b} \in \{0,1\}^t$ be a vector such that $\boldsymbol{b}^T\boldsymbol{V}^* \in [v_i, u_i]$, but $\boldsymbol{b}^T\widehat{\boldsymbol{V}} \notin [v_i, u_i]$. We will prove that $v_i \leqslant \boldsymbol{b}^T\widetilde{\boldsymbol{V}} \leqslant u_i$.

Without loss of generality, we assume $\boldsymbol{b}^T\widehat{\boldsymbol{V}} < v_i \leqslant \boldsymbol{b}^T\boldsymbol{V}^* \leqslant u_i$. Then

$$\begin{aligned}
u_i \geqslant \boldsymbol{b}^T\boldsymbol{V}^* &\overset{①}{\geqslant} \boldsymbol{b}^T\widetilde{\boldsymbol{V}} = \boldsymbol{b}^T\left(\boldsymbol{V}^* - \delta_{\min}(\boldsymbol{V}^* - \widehat{\boldsymbol{V}})\right) \\
&\geqslant \boldsymbol{b}^T\boldsymbol{V}^* - \boldsymbol{b}^T\frac{\boldsymbol{b}^T\boldsymbol{V}^* - v_i}{\boldsymbol{b}^T\boldsymbol{V}^* - \boldsymbol{b}^T\widehat{\boldsymbol{V}}}(\boldsymbol{V}^* - \widehat{\boldsymbol{V}}) \\
&= \boldsymbol{b}^T\boldsymbol{V}^* - \boldsymbol{b}^T(\boldsymbol{V}^* - \widehat{\boldsymbol{V}})\frac{\boldsymbol{b}^T\boldsymbol{V}^* - v_i}{\boldsymbol{b}^T(\boldsymbol{V}^* - \widehat{\boldsymbol{V}})} \\
&= v_i,
\end{aligned}$$

where ① follows from the fact that

$$\begin{aligned}
\boldsymbol{b}^T\widetilde{\boldsymbol{V}} &= \boldsymbol{b}^T\left(\boldsymbol{V}^* - \delta_{\min}(\boldsymbol{V}^* - \widehat{\boldsymbol{V}})\right) \\
&= \boldsymbol{b}^T\boldsymbol{V}^* - \delta_{\min}\boldsymbol{b}^T(\boldsymbol{V}^* - \widehat{\boldsymbol{V}}) \\
&\leqslant \boldsymbol{b}^T\boldsymbol{V}^*.
\end{aligned}$$

If $c_i$ can be programmed into its quantization interval $[v_i, u_i], i \in [n]$ by $\boldsymbol{V}^*$ and $\widehat{\boldsymbol{V}}$, then it can be programmed into $[v_i, u_i]$ by $\widetilde{\boldsymbol{V}}$ as well, since $\widetilde{\boldsymbol{V}}$ is a convex combination of $\boldsymbol{V}^*$ and $\widehat{\boldsymbol{V}}$. Thus, each cell that can be quantized correctly by $\boldsymbol{V}^*$ can also be quantized correctly by $\widetilde{\boldsymbol{V}}$, implying that $\widetilde{\boldsymbol{V}}$ is optimal. ■

Let

$$\widetilde{\mathbf{A}} = \begin{bmatrix} \mathbf{A}' \\ \boldsymbol{b}_{k_{\min}}^T \end{bmatrix}.$$

**Claim 6.8.6.** *$\widetilde{\mathbf{A}}$ has row rank $t'+1$ and $\widetilde{\mathbf{A}}\widetilde{\boldsymbol{V}} \in \mathcal{T}^{(t'+1)}$.*

**Proof.** According to Claim 6.8.4, each $\boldsymbol{b}^T$ is linearly independent of the set of row vectors of $\mathbf{A}'$, implying that $\text{rank}(\widetilde{\mathbf{A}}) = t' + 1$.

Consider

$$\widetilde{\mathbf{A}}\widetilde{\boldsymbol{V}} = \begin{bmatrix} \mathbf{A}' \\ \boldsymbol{b}_{k_{\min}}^T \end{bmatrix} \widetilde{\boldsymbol{V}} = \begin{bmatrix} \mathbf{A}'\widetilde{\boldsymbol{V}} \\ \boldsymbol{b}_{k_{\min}}^T \widetilde{\boldsymbol{V}} \end{bmatrix} \stackrel{\text{def}}{=} \widetilde{\boldsymbol{p}},$$

Since $\widetilde{\boldsymbol{V}}$ is a convex combination of $\boldsymbol{V}^*$ and $\widehat{\boldsymbol{V}}$, it is in the polytope $\mathcal{P}$, thus $\mathbf{A}'\widetilde{\boldsymbol{V}} = \boldsymbol{p}' \in \mathcal{T}^{t'}$.

Now,

$$\begin{aligned}
\boldsymbol{b}_{k_{\min}}^T \widetilde{\boldsymbol{V}} &= \boldsymbol{b}_{k_{\min}}^T \left( \boldsymbol{V}^* - \delta_{\min}(\boldsymbol{V}^* - \widehat{\boldsymbol{V}}) \right) \\
&= \boldsymbol{b}_{k_{\min}}^T \left( \boldsymbol{V}^* - \frac{\boldsymbol{b}_{k_{\min}}^T \boldsymbol{V}^* - w_{k_{\min}}}{\boldsymbol{b}_{k_{\min}}^T \boldsymbol{V}^* - \boldsymbol{b}_{k_{\min}}^T \widehat{\boldsymbol{V}}} (\boldsymbol{V}^* - \widehat{\boldsymbol{V}}) \right) \\
&= \boldsymbol{b}_{k_{\min}}^T \boldsymbol{V}^* - \boldsymbol{b}_{k_{\min}}^T (\boldsymbol{V}^* - \widehat{\boldsymbol{V}}) \frac{\boldsymbol{b}_{k_{\min}}^T \boldsymbol{V}^* - w_{k_{\min}}}{\boldsymbol{b}_{k_{\min}}^T (\boldsymbol{V}^* - \widehat{\boldsymbol{V}})} \\
&= w_{k_{\min}} \in \mathcal{T}.
\end{aligned}$$

Therefore, $\widetilde{\boldsymbol{p}} \in \mathcal{T}^{(t'+1)}$. ■

For both Case (1) and Case (2), the existence of $\widetilde{\mathbf{A}}$, $\widetilde{\boldsymbol{V}}$ and $\widetilde{\boldsymbol{p}}$ contradicts the assumption that $t'$ is the maximum row rank of a matrix $\mathbf{A}$ such that $\mathbf{A}\boldsymbol{V}^* = \boldsymbol{p}'$. Therefore, there exists an invertible matrix $\mathbf{A} \in \{0,1\}^{t \times t}$ such that

$$\mathbf{A}\boldsymbol{V} = \boldsymbol{p},$$

where $\boldsymbol{V} \in \mathbb{R}_+^t$ is an optimal solution for (P3) and $\boldsymbol{p} \in \mathcal{T}^t$ is a threshold-point vector. ■

**Proof of Lemma 6.4.6.** The proof is based on induction and is very similar to the proof of Lemma 6.3.4. Therefore, we prove the initial step of the induction and omit the remaining details.

For $t = 1$, we prove that there exist a threshold-point $p_i \in \mathcal{T}$ of the cell $i$ and a real number

$$\begin{aligned}
a \in \{&0, 1, \beta_{i-1,i}, \beta_{i+1,i}, 1 + \beta_{i-1,i}, 1 + \beta_{i+1,i}, \beta_{i-1,i} + \beta_{i+1,i}, \\
&1 + \beta_{i-1,i} + \beta_{i+1,i}\} \\
= &\bigcup_{\boldsymbol{b} \in \{0,1\}^3} \{(\beta_{i-1,i}, 1, \beta_{i+1,i}) \cdot \boldsymbol{b}\}
\end{aligned}$$

such that

$$aV = p_i,$$

where $V \geqslant 0$ is optimal.

Suppose $V^*$ is optimal. If $\exists a \in \{0, 1, \beta_{i-1,i}, \beta_{i+1,i}, 1+\beta_{i-1,i}, 1+\beta_{i+1,i}, \beta_{i-1,i}+\beta_{i+1,i}, 1+\beta_{i-1,i}+\beta_{i+1,i}\}$ and $p_i \in \mathcal{T}$ such that

$$aV^* = p_i,$$

then the statement holds for $t = 1$. Otherwise, let

$$\delta = \min_{i\in[n],\boldsymbol{b}\in\{0,1\}^3} \left(V^*(\beta_{i-1,i}, 1, \beta_{i+1,i}) \cdot \boldsymbol{b} - p_i\right)^+,$$

where, for $x \in \mathbb{R}$, $x^+ = x$ if $x \geqslant 0$ and $x = +\infty$ if $x < 0$. Suppose the minimum is achieved for $i = i_{\min} \in [n]$ and $\boldsymbol{b} = \boldsymbol{b}_{\min}$. That is,

$$\delta = (\beta_{i_{\min}-1,i}, 1, \beta_{i_{\min}+1,i}) \cdot \boldsymbol{b}_{\min} V^* - p_{i_{\min}}.$$

Let

$$\widehat{V} = \frac{p_{i_{\min}}}{p_{i_{\min}} + \delta} V^*.$$

Then, by setting $a = (\beta_{i_{\min}-1,i}, 1, \beta_{i_{\min}+1,i}) \cdot \boldsymbol{b}_{\min}$, we have

$$\begin{aligned} a\widehat{V} &= (\beta_{i_{\min}-1,i}, 1, \beta_{i_{\min}+1,i}) \cdot \boldsymbol{b}_{\min} \widehat{V} \\ &= (\beta_{i_{\min}-1,i}, 1, \beta_{i_{\min}+1,i}) \cdot \boldsymbol{b}_{\min} \frac{p_{i_{\min}}}{p_{i_{\min}} + \delta} V^* \\ &= p_{i_{\min}}. \end{aligned}$$

In addition, the number of cells quantized correctly does not decrease in going from $V^*$ to $\widehat{V}$ since, by definition, $\widehat{V} \leqslant V^*$ is chosen such that if a cell is quantized correctly by the voltage $V^*$, the cell can be also quantized correctly by $\widehat{V}$. This completes the proof of the case where $t = 1$. ■

## 6.9 Appendix B

**Proof of Lemma 6.5.2.** We first prove the following claim as it is used to establish the inequalities in the proof of Lemma 6.5.2.

**Claim 6.9.1.** *Let $c_1, c_2$ and $\delta_1, \delta_2 > 0$ be real numbers. Let*

$$p_1 = \frac{1}{\sqrt{2\pi}} \int_{c_1-\delta_1}^{c_1+\delta_1} e^{-u^2/2} \mathrm{d}u \text{ and } p_2 = \frac{1}{\sqrt{2\pi}} \int_{c_2-\delta_2}^{c_2+\delta_2} e^{-u^2/2} \mathrm{d}u.$$

*Then the following three statements hold:*

1. *If $c_1 = c_2$ and $\delta_1 > \delta_2$, then $p_1 > p_2$.*
2. *If $\delta_1 = \delta_2$ and $|c_1| < |c_2|$, then $p_1 > p_2$.*
3. *If $|c_1| \leqslant |c_2|$ and $\delta_1 > \delta_2$, then $p_1 > p_2$.*

**Proof.**

1. If $\delta_1 > \delta_2$, then $[c_1 - \delta_1, c_1 + \delta_1] \supset [c_2 - \delta_2, c_2 + \delta_2]$; thus $p_1 > p_2$ since the integrand is strictly positive on $\mathbb{R}$.

2. Let $\delta_1 = \delta_2 = \delta$. We prove the result for the case where $0 \leqslant c_1 < c_2$ and $[c_1 - \delta, c_1 + \delta] \bigcap [c_2 - \delta, c_2 + \delta] = \emptyset$. Other cases can be reduced to this case by subtracting the integration over the intersection of the intervals. In the case considered, $\xi_1 \leqslant \xi_2, \forall \xi_1 \in [c_1 - \delta, c_1 + \delta], \xi_2 \in [c_2 - \delta, c_2 + \delta]$.

   Note that $f(u) = e^{-u^2/2}$ is symmetric with respect to $u = 0$ and $f(u)$ is a continuous strictly decreasing function for $u \geqslant 0$. Therefore, there exists $\xi_i \in [c_i - \delta, c_i + \delta]$ such that

   $$p_i = \int_{c_i-\delta}^{c_i+\delta} e^{-u^2/2} \mathrm{d}u = 2\delta e^{-\xi_i^2/2}, i = 1, 2.$$

   It follows that $p_1 > p_2$ since $\xi_1 < \xi_2$.

3. Let $p_3 = \int_{c_3-\delta_3}^{c_3+\delta_3} e^{-u^2/2} \mathrm{d}u$, where $c_3 = c_1$ and $\delta_3 = \delta_2$. Then from 1) we have $p_1 \geqslant p_3$, and from 2) we have $p_3 > p_2$. Therefore, $p_1 > p_2$.

■

Now we proceed to the proof of Lemma 6.5.2. Suppose the opposite is true; that is, for some $i$ and $j$ in $[t]$, $V_i^* \neq V_j^*$. Consider another vector $\hat{\boldsymbol{V}} = (\hat{V}_1, \ldots, \hat{V}_t)$, where $\hat{V}_k = \overline{V} \stackrel{\text{def}}{=} \sqrt{\frac{\sum_{j=1}^t V_j^{*2}}{t}}, \forall k \in [t]$. Then $\sqrt{\sum_{j=1}^t \hat{V}_j^2} = \sqrt{\sum_{j=1}^t V_j^{*2}}$. Furthermore, we have

$$\sum_{j=1}^t \hat{V}_j = t\overline{V} = t\sqrt{\frac{\sum_{j=1}^t V_j^{*2}}{t}} > \sum_{j=1}^t V_j^*.$$

This is a special case of the Cauchy-Schwartz Inequality, and the inequality is strict due to the assumption that $V_i^* \neq V_j^*$ for some $i, j \in [t]$. Therefore, $\theta - \alpha \sum_{j=1}^t \hat{V}_j < \theta - \alpha \sum_{j=1}^t V_j^*$.

We want to show that $\boldsymbol{V}^*$ is not optimal, in particular, $g(\hat{\boldsymbol{V}}) > g(\boldsymbol{V}^*)$. We consider two cases.

Case (1): Suppose $\theta - \alpha \sum_{j=1}^t \hat{V}_j = \theta - \alpha t \overline{V} \geqslant 0$.

Let

$$p_1 = g(\boldsymbol{V}^*) = \frac{1}{\sqrt{2\pi}} \int_{c(\boldsymbol{V}^*)-\delta(\boldsymbol{V}^*)}^{c(\boldsymbol{V}^*)+\delta(\boldsymbol{V}^*)} e^{-\frac{u^2}{2}} \mathrm{d}u$$

and

$$p_2 = g(\hat{\boldsymbol{V}}) = \frac{1}{\sqrt{2\pi}} \int_{c(\hat{\boldsymbol{V}})-\delta(\hat{\boldsymbol{V}})}^{c(\hat{\boldsymbol{V}})+\delta(\hat{\boldsymbol{V}})} e^{-\frac{u^2}{2}} \mathrm{d}u,$$

where $c(\boldsymbol{V})$ and $\delta(\boldsymbol{V})$ are as defined in Lemma 6.5.1.

Recall that $c(\hat{\boldsymbol{V}}) = \frac{\theta - \alpha \sum \hat{V}_j}{\sigma \sqrt{\sum \hat{V}_j^2}}$ and $c(\boldsymbol{V}^*) = \frac{\theta - \alpha \sum V_j^{*2}}{\sigma \sqrt{\sum V_j^{*2}}}$, and let $s = \sqrt{\sum_{j=1}^t \hat{V}_j^2} = \sqrt{\sum_{j=1}^t V_j^{*2}}$. Then we have

$$\begin{aligned}
0 \leqslant c(\hat{\boldsymbol{V}}) &= \frac{\theta - \alpha \sum_{j=1}^t \hat{V}_j}{\sigma \sqrt{\sum_{j=1}^t \hat{V}_j^2}} \\
&= \frac{\theta - \alpha \sum_{j=1}^t \hat{V}_j}{s\sigma} \\
&< \frac{\theta - \alpha \sum_{j=1}^t V_j^{*2}}{s\sigma} \\
&= \frac{\theta - \alpha \sum_{j=1}^t V_j^{*2}}{\sigma \sqrt{\sum_{j=1}^t V_j^{*2}}} \\
&= c(\boldsymbol{V}^*).
\end{aligned}$$

Recall that $\delta(\hat{\boldsymbol{V}}) = \frac{\Delta}{\sigma \sqrt{\sum \hat{V}_j^2}}$ and $\delta(\boldsymbol{V}^*) = \frac{\Delta}{\sigma \sqrt{\sum V_j^{*2}}}$. Then $\delta(\hat{\boldsymbol{V}}) = \delta(\boldsymbol{V}^*)$. Set $c_1 = c(\hat{\boldsymbol{V}}), c_2 = c(\boldsymbol{V}^*), \delta_1 = \delta(\hat{\boldsymbol{V}})$, and $\delta_2 = \delta(\boldsymbol{V}^*)$. According to 2) in Claim 6.9.1, we conclude that $p_1 < p_2$. Therefore, $g(\boldsymbol{V}^*) < g(\hat{\boldsymbol{V}})$, implying $\boldsymbol{V}^*$ is not optimal.

Case (2): Suppose $\theta - \alpha \sum_{j=1}^t \hat{V}_j = \theta - \alpha t \overline{V} < 0$.

Consider another vector $\tilde{\boldsymbol{V}} = (\tilde{V}_1, \ldots, \tilde{V}_t)$ where $\tilde{V}_j = \frac{\theta}{\alpha t}, \forall j \in [t]$. Then $\theta - \alpha \sum_{j=1}^t \tilde{V}_j = 0$ and we have

$$\sqrt{\sum_{j=1}^t \tilde{V}_j^2} = \sqrt{\left(\frac{\theta}{\alpha t}\right)^2 t} < \sqrt{\overline{V}^2 t} \qquad = \sqrt{\sum_{j=1}^t \hat{V}_j^2} = \sqrt{\sum_{j=1}^t V_j^{*2}}.$$

It is easy to see that

$$|c(\tilde{\boldsymbol{V}})| = \Big|\frac{\theta - \alpha\sum_{j=1}^{t}\tilde{V}_j}{\sigma\sqrt{\sum_{j=1}^{t}\tilde{V}_j^2}}\Big| = 0 \leqslant |c(\boldsymbol{V}^*)|,$$

and

$$\delta(\tilde{\boldsymbol{V}}) = \frac{\Delta}{\sigma\sqrt{\sum_{j=1}^{t}\tilde{V}_j^2}} > \frac{\Delta}{\sigma\sqrt{\sum_{j=1}^{t}V_j^{*2}}} = \delta(\boldsymbol{V}^*).$$

Let $p_1 = g(\tilde{\boldsymbol{V}})$ and $p_2 = g(\boldsymbol{V}^*)$. Set $c_1 = c(\tilde{\boldsymbol{V}}), c_2 = c(\boldsymbol{V}^*), \delta_1 = \delta(\tilde{\boldsymbol{V}})$, and $\delta_2 = \delta(\boldsymbol{V}^*)$. According to 3) in Claim 6.9.1, we conclude that $p_1 > p_2$. Therefore, $g(\tilde{\boldsymbol{V}}) > g(\boldsymbol{V}^*)$, implying that $\boldsymbol{V}^*$ is not optimal.

These contradictions of the optimality of $\boldsymbol{V}^*$ arose from the assumption that $V_i^* \neq V_j^*$ for some $i, j \in [t]$. Therefore, we conclude that $V_i^* = V_j^*, \forall i, j \in [t]$. ■

## 6.10 Appendix C

**Proof of Lemma 6.6.2.** Recall that when applying voltage $V_j$, the cell-level increment is uniformly distributed in $[(\alpha - \delta_1)V_j, (\alpha + \delta_2)V_j]$. Consider the following two cases.

Case (1): Suppose $\frac{\theta-\Delta}{\theta+\Delta} < \frac{\alpha-\delta_1}{\alpha+\delta_2}$. Setting $V_1 = \frac{\theta+\Delta}{\alpha+\delta_2}$, we have

$$\begin{aligned}
P(V_1, \theta, \Delta, 1) &= \int_{\theta-\Delta}^{\theta+\Delta} p_1(x - \alpha V_1)\mathrm{d}x \\
&= \int_{\theta-\Delta}^{\theta+\Delta} \frac{1}{(\delta_1+\delta_2)V_1} I_{x-\alpha V_1 \in [-\delta_1 V_1, \delta_2 V_1]}\mathrm{d}x \\
&= \int_{\theta-\Delta}^{\theta+\Delta} \frac{1}{(\delta_1+\delta_2)V_1} I_{x \in [(\alpha-\delta_1)V_1, (\alpha+\delta_2)V_1]}\mathrm{d}x \\
&= \int_{(\alpha-\delta_1)V_1}^{(\alpha+\delta_2)V_1} \frac{1}{(\delta_1+\delta_2)V_1}\mathrm{d}x \\
&= 1.
\end{aligned}$$

Therefore,

$$1 \geqslant P(\theta, \Delta, 1) = \max_{V_1} P(V_1, \theta, \Delta, 1) \geqslant 1.$$

Case (2): Suppose $\frac{\theta-\Delta}{\theta+\Delta} \geqslant \frac{\alpha-\delta_1}{\alpha+\delta_2}$. Then

$$\begin{aligned} P(V_1,\theta,\Delta,1) &= \int_{\theta-\Delta}^{\theta+\Delta} p_1(x-\alpha V_1)\mathrm{d}x \\ &= \int_{\theta-\Delta}^{\theta+\Delta} \frac{1}{(\delta_1+\delta_2)V_1} I_{x-\alpha V_1 \in [-\delta_1 V_1, \delta_2 V_1]}\mathrm{d}x \\ &= \int_{\mathbb{R}} \frac{1}{(\delta_1+\delta_2)V_1} I_{x\in[(\alpha-\delta_1)V_1,(\alpha+\delta_2)V_1]\bigcap[\theta-\Delta,\theta+\Delta]}\mathrm{d}x. \end{aligned}$$

There are two possibilities to consider.

1. If $V_1 \leqslant \frac{\theta+\Delta}{\alpha+\delta_2}$, then $(\alpha+\delta_2)V_1 \leqslant \theta+\Delta$ and $(\alpha-\delta_1)V_1 \leqslant \theta-\Delta$. Therefore

$$\begin{aligned} &P(V_1,\theta,\Delta,1) \\ &= \int_{\mathbb{R}} \frac{I_{x\in[(\alpha-\delta_1)V_1,(\alpha+\delta_2)V_1]\bigcap[\theta-\Delta,\theta+\Delta]}}{(\delta_1+\delta_2)V_1}\mathrm{d}x \\ &= \int_{\theta-\Delta}^{(\alpha+\delta_2)V_1} \frac{1}{(\delta_1+\delta_2)V_1}\mathrm{d}x \\ &= \frac{\alpha+\delta_2}{\delta_1+\delta_2} - \frac{\theta-\Delta}{(\delta_1+\delta_2)V_1} \\ &\leqslant \frac{\alpha+\delta_2}{\delta_1+\delta_2} - \frac{\theta-\Delta}{(\delta_1+\delta_2)\frac{\theta+\Delta}{\alpha+\delta_2}} \\ &= \left(\frac{\alpha+\delta_2}{\delta_1+\delta_2}\right)\left(\frac{2\Delta}{\theta+\Delta}\right), \end{aligned}$$

   where equality holds if $V_1 = \frac{\theta+\Delta}{\alpha+\delta_2}$.

2. If $V_1 > \frac{\theta+\Delta}{\alpha+\delta_2}$, then $(\alpha+\delta_2)V_1 > \theta+\Delta$. Therefore

$$\begin{aligned} &P(V_1,\theta,\Delta,1) \\ &= \int_{\mathbb{R}} \frac{I_{x\in[(\alpha-\delta_1)V_1,(\alpha+\delta_2)V_1]\bigcap[\theta-\Delta,\theta+\Delta]}}{(\delta_1+\delta_2)V_1}\mathrm{d}x \\ &\leqslant \int_{\theta-\Delta}^{\theta+\Delta} \frac{1}{(\delta_1+\delta_2)V_1}\mathrm{d}x \\ &= \frac{2\Delta}{(\delta_1+\delta_2)V_1} \\ &< \frac{2\Delta}{(\delta_1+\delta_2)\frac{\theta+\Delta}{\alpha+\delta_2}} \\ &= P\left(\frac{\theta+\Delta}{\alpha+\delta_2},\theta,\Delta,1\right) \end{aligned}$$

It can be seen that under both circumstances, $P(V_1, \theta, \Delta, 1)$ is maximized when $V_1 = \frac{\theta+\Delta}{\alpha+\delta_2}$. Consequently,

$$P(\theta, \Delta, 1) = \begin{cases} 1, & \text{if } \frac{\theta-\Delta}{\theta+\Delta} < \frac{\alpha-\delta_1}{\alpha+\delta_2}, \\ \frac{\alpha+\delta_2}{\delta_1+\delta_2}\frac{2\Delta}{\theta+\Delta} & \text{if } \frac{\theta-\Delta}{\theta+\Delta} \geqslant \frac{\alpha-\delta_1}{\alpha+\delta_2}. \end{cases}$$

■

**Proof of Lemma 6.6.3.** We first prove the following two claims as they serve as the basis for the proof of Lemma 6.6.3.

**Claim 6.10.1.** *For $\beta \neq 0$, $P(\beta\theta, \beta\Delta, t) = P(\theta, \Delta, t)$.*

**Proof.** We proceed by induction. For $t = 1$, we have

$$\begin{aligned} P(\beta\theta, \beta\Delta, 1) &= \begin{cases} 1, & \text{if } \frac{\beta\theta-\beta\Delta}{\beta\theta+\beta\Delta} < \frac{\alpha-\delta_1}{\alpha+\delta_2}, \\ \frac{\alpha+\delta_2}{\delta_1+\delta_2}\frac{2\beta\Delta}{\beta\theta+\beta\Delta} & \text{if } \frac{\beta\theta-\beta\Delta}{\beta\theta+\beta\Delta} \geqslant \frac{\alpha-\delta_1}{\alpha+\delta_2}, \end{cases} \\ &= \begin{cases} 1, & \text{if } \frac{\theta-\Delta}{\theta+\Delta} < \frac{\alpha-\delta_1}{\alpha+\delta_2}, \\ \frac{\alpha+\delta_2}{\delta_1+\delta_2}\frac{2\Delta}{\theta+\Delta} & \text{if } \frac{\theta-\Delta}{\theta+\Delta} \geqslant \frac{\alpha-\delta_1}{\alpha+\delta_2}, \end{cases} \\ &= P(\theta, \Delta, 1). \end{aligned}$$

For notational convenience, let $a \bigwedge b \stackrel{\text{def}}{=} \min\{a, b\}$. Now, suppose that $P(\beta\theta, \beta\Delta, t-1) = P(\theta, \Delta, t-1)$. Then

$$
\begin{aligned}
&P(\beta\theta, \beta\Delta, t) \\
=&\max_{V_1} \int_{\mathbb{R}_+} p_1(x - \alpha V_1) P(\beta\theta - x, \beta\Delta, t-1) \mathrm{d}x \\
\overset{(1)}{=}&\max_{V_1} \int_{\mathbb{R}_+} \frac{I_{x \in [(\alpha-\delta_1)V_1, ((\alpha+\delta_2)V_1) \wedge \beta\theta]}}{(\delta_1+\delta_2)V_1} \\
&\qquad \cdot P(\beta\theta - x, \beta\Delta, t-1) \mathrm{d}x \\
\overset{(2)}{=}&\max_{V_1} \int_{\mathbb{R}_+} \frac{I_{\beta y \in [(\alpha-\delta_1)V_1, ((\alpha+\delta_2)V_1) \wedge \beta\theta]}}{(\delta_1+\delta_2)V_1} \\
&\qquad \cdot P(\beta\theta - \beta y, \beta\Delta, t-1) \mathrm{d}\beta y \\
\overset{(3)}{=}&\max_{V_1} \int_{\mathbb{R}_+} \frac{I_{\beta y \in [(\alpha-\delta_1)V_1, ((\alpha+\delta_2)V_1) \wedge \beta\theta]}}{(\delta_1+\delta_2)V_1} \\
&\qquad \cdot P(\theta - y, \Delta, t-1) \mathrm{d}\beta y \\
\overset{(4)}{=}&\max_{V_1} \int_{\mathbb{R}_+} \frac{I_{y \in [(\alpha-\delta_1)\frac{V_1}{\beta}, \left((\alpha+\delta_2)\frac{V_1}{\beta}\right) \wedge \theta]}}{(\delta_1+\delta_2)\frac{V_1}{\beta}} P(\theta - y, \Delta, t-1) \mathrm{d}y \\
\overset{(5)}{=}&\max_{V_1'} \int_{\mathbb{R}_+} \frac{I_{y \in [(\alpha-\delta_1)V_1', \left((\alpha+\delta_2)V_1'\right) \wedge \theta]}}{(\delta_1+\delta_2)V_1'} P(\theta - y, \Delta, t-1) \mathrm{d}y \\
\overset{(6)}{=}&\max_{V_1'} \int_{\mathbb{R}_+} p_1(y - \alpha V_1') P(\theta - y, \Delta, t-1) \mathrm{d}y \\
=&P(\theta, \Delta, t).
\end{aligned}
$$

Equation $(1)$ follows from the definition of $p_1(x)$. Equation $(2)$ follows from the change of variables $x = \beta y$. Equation $(3)$ follows from the induction hypothesis that $P(\beta\theta, \beta\Delta, t-1) = P(\theta, \Delta, t-1)$. Equation $(4)$ follows from the linearity of the indicator function and the min operator. That is $I_{\beta y \in [a,b]} = I_{y \in [a/\beta, b/\beta]}$, and $\frac{1}{y}\min\{a, b\} = \min\{\frac{a}{y}, \frac{b}{y}\}$. Equation $(5)$ follows from the change of variables $V_1' = \frac{V_1}{\beta}$. Equation $(6)$ holds for the same reason as Equation $(1)$. ■

**Claim 6.10.2.** *For $\beta > 1$, $P(\beta\theta, \beta\Delta, t) \geqslant P(\beta\theta, \Delta, t)$.*

**Proof.** Since the interval $[\beta\theta - \Delta, \beta\theta + \Delta] \subseteq [\beta\theta - \beta\Delta, \beta\theta + \beta\Delta]$ for $\beta > 1$, the event "the cell level is in $[\beta\theta - \Delta, \beta\theta + \Delta]$ after $t$ rounds of programming" is included in the event "the cell level is in $[\beta\theta - \beta\Delta, \beta\theta + \beta\Delta]$ after $t$ rounds of programming". Thus, $P(\beta\theta, \beta\Delta, t) \geqslant P(\beta\theta, \Delta, t)$. ■

It follows from Claim 6.10.1 and Claim 6.10.2 that, for $\beta > 1$,

$$P(\theta, \Delta, t) = P(\beta\theta, \beta\Delta, t) \geqslant P(\beta\theta, \Delta, t),$$

which proves that $P(\theta, \Delta, t)$ is non-increasing in $\theta$. ■

## Acknowledgments

This chapter is in part a reprint of the material in the paper: Minghai Qin, Eitan Yaakobi, and Paul H. Siegel, "Optimized cell programming for flash memories with quantizers", IEEE Transaction on Information Theory, vol. 60, no. 5, pp. 1-16, May 2014.

I would also like to thank Lele Wang for her comments on the statement and proof of Lemma 6.3.4.

# Chapter 7

# Parallel programming of rank modulation for flash memories

## 7.1 Introduction

One way in which the storage capacity of flash memory devices has been increased is through the use of a larger number of levels in each cell. However, as the quantization of the cell levels becomes finer, the problem of overshooting can be exacerbated, thereby compromising data integrity and limiting the achievable storage capacity. A novel idea called **rank modulation** [48] has been proposed to solve such overshooting problems. With rank modulation, the information is represented by the permutation induced by the levels of an ordered set of $n$ cells. As a result, it does not require a discrete set of target cell levels, and recovery of the stored information is easily implemented by comparing the charges of the $n$ cells. However, to date, programming algorithms for rank modulation, as described in [32, 48, 49], for example, have been based upon sequential programming of individual cells, resulting in fundamental limitations on the programming speed.

In this chapter, we study two approaches to parallel programming for rank modulation where cell levels are integer-valued. In the first, our objective is to minimize the number of programming rounds needed to produce a cell-level vector representing the target permutation, under the assumption that there is no constraint on the magnitude of cell-level increments. A consequence of this assumption is that the cell levels may approach their physical upper limits quickly, implying that this programming technique may limit the number of information updates that can be made before a block erasure is required. In contrast, the second approach aims to

minimize the number of programming rounds subject to a constraint on the difference between the maximum cell level before and after programming. This technique therefore allows the maximum possible number of subsequent information updates before a block erasure is required. These two scenarios represent different trade-offs between programming speed and the lifetime of the flash memory.

Since the minimum number of programming rounds is a function of the target permutation $\tau$ and the initial cell-state vector $\boldsymbol{\ell}_0$, we denote them by $t_1^*(\tau, \boldsymbol{\ell}_0)$ and $t_2^*(\tau, \boldsymbol{\ell}_0)$, respectively, in the two programming scenarios. We will use the notation $t_i^*$ and $t_i^*(\tau, \boldsymbol{\ell}_0)$ interchangeably, $i = 1, 2$, in contexts where there is no ambiguity. We first derive universal lower and upper bounds on $t_1^*(\tau, \boldsymbol{\ell}_0)$ as a function of $\tau$. These bounds show that, when compared to the push-to-top programming scheme proposed in [48], the minimal number of programming rounds required in this scenario is approximately $\frac{1}{3}$ of that used in the push-to-top scheme. In the second scenario, the smallest set of cell-level increments needed to represent the target permutation can be determined as in [32]. We derive an upper bound on $t_2^*(\tau, \boldsymbol{\ell}_0)$ and propose an algorithm with complexity $O(n^2 \log n)$ to search for the optimal set of voltages in each round of programming. The resulting number of programming rounds is shown to be only 2.5% more than the minimum number obtained by a brute-force optimal search.

The rest of the chapter is organized as follows. In Section 7.2, we model the cell increments during parallel programming mathematically. In Section 7.3, we aim to minimize the number of programming rounds for rank modulation when cells are allowed to increase to arbitrarily high levels. In Section 7.4, we aim to minimize the number of programming rounds for rank modulation when the cell increment is minimized.

## 7.2 Preliminaries

We denote $n$ flash memory cells by $c_1, \ldots, c_n$. Cells are programmed by injecting charge to increase the cell levels. We denote by $[m : n]$ the set of integers $\{i \in \mathbb{Z} | m \leqslant i \leqslant n\}$ and $[1 : n]$ is denoted by $[n]$. The set of non-negative real numbers and non-negative integers are denoted by $\mathbb{R}_+$ and $\mathbb{Z}_+$, respectively. As assumed in [70,94], the number of programming rounds (charge injection) is $t$ and when a voltage $V_j$, $j \in [t]$, is applied to cell $c_i$, the cell level of $c_i$ will be increased by $\alpha_i V_j$. We call $\alpha_i$ the hardness of charge injection of $c_i$, and we define the cell-hardness vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$. Suppose the cell-state vector is $\boldsymbol{\ell}_0$ before programming and denote the set of voltages that can be applied in the $t$ rounds of parallel programming by $\boldsymbol{V} = (V_1, \ldots, V_t)$. The final cell-state vector is then

$$\ell_{i,t} = \ell_{i,0} + \alpha_i \sum_{j=1}^{t} b_{i,j} V_j, \tag{7.1}$$

where the $(i, j)$ entry in the matrix

$$\mathbf{B} = \begin{bmatrix} b_{1,1} & b_{2,1} & \cdots & b_{n,1} \\ b_{1,2} & b_{2,2} & \cdots & b_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ b_{1,t} & b_{2,t} & \cdots & b_{n,t} \end{bmatrix} \in \{0,1\}^{t \times n}$$

indicates whether or not voltage $V_j$ is applied to cell $c_i$ during the $j$-th round of programming, for $i \in [n], j \in [t]$.

Let $\boldsymbol{\ell}_t = (\ell_{1,t}, \ldots, \ell_{n,t})$ be the final cell-state vector after programming. We assume $\ell_{i,t} \neq \ell_{j,t}, \forall i \neq j$. Let $\Sigma_n$ be the set of all permutations of $[n]$. Let $\tau = (\tau_1, \ldots, \tau_n) \in \Sigma_n$ be the rank permutation of $\boldsymbol{\ell}_t$, denoted by $\text{rank}(\boldsymbol{\ell}_t) = \tau$, where $\tau_i$ is the index of the cell with the $i$-th lowest level. For example, if $\boldsymbol{\ell}_t = (1.5, 3.5, 0.5, 2)$, then $\text{rank}(\boldsymbol{\ell}_t) = \tau = (3, 1, 4, 2)$.

The figure of merit for the programming algorithm is the number of required programming rounds, $t$. The cell programming problem is to find a set of positive real programming voltages $\boldsymbol{V} \in \mathbb{R}_+^t$ and a binary matrix $\mathbf{B} \in \{0,1\}^{t \times n}$ of cell-programming indicators such that $\text{rank}(\boldsymbol{\ell}_t) = \tau$ and $t$ is minimized, where $\boldsymbol{\ell}_t$ is given by Equation (7.1).

## 7.3 Rank modulation minimizing programming rounds

In this section, we assume $\alpha_i = 1, \forall i \in [n]$ and both $\ell_{i,j}$ and $V_j, i \in [n], j \in [t]$ are integers. We further assume there is no physical upper limit on the cell levels. Given the initial cell-state vector $\boldsymbol{\ell}_0$ with $\text{rank}(\boldsymbol{\ell}_0) \in \Sigma_n$ and the target permutation $\tau \in \Sigma_n$, we would like to find a set of programming voltages $\boldsymbol{V} \in \mathbb{Z}_+^t$ and a binary matrix $\mathbf{B} \in \{0,1\}^{t \times n}$ of cell-programming indicators such that $\text{rank}(\boldsymbol{\ell}_t) = \tau$ and $t$ is minimized, where $\boldsymbol{\ell}_t$ can now be expressed as

$$\ell_{i,t} = \ell_{i,0} + \sum_{j=1}^{t} b_{i,j} V_j. \tag{7.2}$$

The following definitions will be useful in deriving bounds on the minimum possible value of $t$.

**Definition 7.3.1.** *A **decomposition** of a permutation $\sigma = (\sigma_1, \ldots, \sigma_n) \in \Sigma_n$ decomposes $\sigma$ into subsequences $((v_1), \ldots, (v_m))$, where $(v_i), i \in [m]$, is a subsequence of the form $(\sigma_{u_1}, \sigma_{u_2}, \ldots, \sigma_{u_k})$, in which the relative orders within $\sigma$ are preserved, i.e., for any two $u_j, u_k \in [n]$ in $(v_i)$, $\sigma_{u_j}$ is to the left of $\sigma_{u_k}$ in $v_i$ if and only if $\sigma_{u_j}$ is to the left of $\sigma_{u_k}$ in $\sigma$.*

*An **increasing block** $(\sigma_u, \sigma_{u+1}, \ldots, \sigma_v), 1 \leqslant u \leqslant v \leqslant n$, of a permutation $\sigma = (\sigma_1, \ldots, \sigma_n)$ is a contiguous subsequence of $\sigma$ such that $\sigma_i < \sigma_{i+1}, \forall i \in [u : v-1]$. An increasing block decomposition of $\sigma$ is a decomposition such that each subsequence is an increasing block.*

*An **increasing subsequence** $(\sigma_{u_1}, \sigma_{u_2}, \ldots, \sigma_{u_k}), u_i \in [n], i \in [k]$, of a permutation $\sigma = (\sigma_1, \ldots, \sigma_n)$ is a subsequence (not necessarily contiguous) of $\sigma$ such that $\sigma_{u_i} < \sigma_{u_{i+1}}, \forall i \in [k-1]$. An increasing subsequence decomposition of $\sigma$ is a decomposition such that each subsequence is an increasing subsequence. A **decreasing subsequence** is defined similarly.*

**Example 7.3.1.** If $\sigma = (3, 1, 4, 5, 6, 2)$, then an increasing block decomposition is $((3), (1, 4, 5, 6), (2))$ and an increasing subsequence decomposition is $((3, 4, 5, 6), (1, 2))$.

Without loss of generality, we assume the initial rank is the identity permutation, i.e., $\sigma = (1, 2, \ldots, n)$. It is clear that the optimal $t^*$ depends on $\tau$ and the initial cell-state vector $\boldsymbol{\ell}_0$, so we denote it by $t_1^*(\tau, \boldsymbol{\ell}_0)$. The next theorem gives a universal bound on $t_1^*(\tau, \boldsymbol{\ell}_0)$ as a function of $\tau$.

**Theorem 7.3.2.** *Let $m_1$ and $m_2$ be the minimum number of subsequences in an increasing block decomposition and an increasing subsequence decomposition of $\tau$, respectively. Then the optimal $t_1^*$ satisfies*

$$\lceil \log m_2 \rceil \leqslant t_1^*(\tau, \boldsymbol{\ell}_0) \leqslant \lceil \log m_1 \rceil.$$

**Proof.** First we prove the lower bound. We will invoke the following lemma, whose proof is omitted due to space constraints.

**Lemma 7.3.3.** *The minimum number of subsequences in any increasing subsequence decomposition of a permutation $\sigma$ is equal to the length of the longest decreasing subsequence in $\sigma$.*

According to Lemma 7.3.3, the length of the longest decreasing subsequence in $\tau$ is $m_2$. Let this decreasing subsequence be

$$(d_1, d_2, \ldots, d_{m_2}),$$

where

$$d_1 > d_2 > \cdots > d_{m_2}$$

and

$$\ell_{d_1,t} < \ell_{d_2,t} < \cdots < \ell_{d_{m_2},t}.$$

Since the initial permutation is assumed to be the identity permutation $\sigma = (1, \ldots, n)$, meaning $\ell_{d_1,0} > \ell_{d_2,0} > \cdots > \ell_{d_{m_2},0}$, the increments of the $d_i$-th cell, denoted by $(I_{d_1}, \ldots, I_{d_{m_2}})$, where $I_{d_i} = \ell_{d_i,t} - \ell_{d_i,0}, \forall i \in [m]$, have to satisfy

$$I_{d_1} < I_{d_2} < \cdots < I_{d_{m_2}},$$

i.e., at least $m_2$ distinct increments are needed. It follows that the number of programming rounds satisfies the lower bound $\lceil \log m_2 \rceil \leqslant t_1^*$.

We now proceed to the proof of the upper bound, which is achieved by a specific programming strategy.

Suppose

$$((\tau_1, \ldots, \tau_{k_1}), (\tau_{k_1+1}, \ldots, \tau_{k_2}), \ldots, (\tau_{k_{m_1-1}+1}, \ldots, \tau_{k_{m_1}}))$$

is an increasing block decomposition of $\tau$ of size $m_1$, where $k_{m_1} = n$. By assumption, the initial permutation is an identity permutation, so the cell levels satisfy

$$\ell_{1,0} < \ell_{2,0} < \cdots < \ell_{n,0} < N$$

for some integer $N$, which implies that $|\ell_{i,0} - \ell_{j,0}| < N, \forall i, j \in [n]$. Let $t = \lceil \log m_1 \rceil$ and $\boldsymbol{V} = (V_1, \ldots, V_t)$, where $V_j = 2^{j-1}N, \forall j \in [t]$. The set of achievable increments is, therefore, $\mathcal{I} = \{iN | i \in [0 : 2^t - 1]\}$. Now, for every index $i \in [n]$, we determine the the block index $j \in [m_1]$ such that $i$ lies in the $j$-th block of the increasing block decomposition of $\tau$, and we increase the level of the cell $c_i$ by $(j-1)N$. Since $1 \leqslant j \leqslant m_1 \leqslant 2^t$, it follows that $(j-1)N \in \mathcal{I}$, meaning that the cell level increments that are produced can all be achieved by $\boldsymbol{V}$. We now show that the rank permutation of the cell-state vector $\boldsymbol{\ell}_t$ attained by this programming strategy is $\tau$.

Consider a pair of cell indices $u$ and $v$ in $\tau$ such that $u$ is to the left of $v$. If they are both in the same increasing block, then

$$\ell_{u,t} = \ell_{u,0} + I_u = \ell_{u,0} + I_v < \ell_{v,0} + I_v = \ell_{v,t},$$

where the first and last equalities follow from the definition of $I_u, u \in [n]$, and the second equality follows from the fact that the cell increments are the same within each block and the inequality follows from the relation $u < v$. If $u$ is in the $i$-th block and $v$ is in the $j$-th block, where $i < j$, then

$$\begin{aligned} \ell_{u,t} &= \ell_{u,0} + I_u = \ell_{u,0} + (i-1)N \\ &< N + (i-1)N = iN \leqslant (j-1)N = I_v < \ell_{v,t}. \end{aligned}$$

This proves $\ell_{u,t} < \ell_{v,t}$ if $u$ is on the left of $v$ in $\tau$. Since $u$ and $v$ are chosen arbitrarily, the rank permutation of the final cell-state vector $\boldsymbol{\ell}_t$ is $\tau$. Therefore, $t_1^* \leqslant \lceil \log m_1 \rceil$. ■

Lemma 7.3.3 establishes the connection between the increasing subsequence decomposition and the longest decreasing subsequence of a permutation $\tau \in \Sigma_n$. Efficient algorithms with time complexity $O(n \log n)$ that find the longest increasing/decreasing subsequence of $\tau \in \Sigma_n$ were studied in [74, 78, 79] and it was shown that partitioning a permutation into a minimum number of monotone subsequences is NP-hard [79]. The problem of decomposing a permutation into a minimum number of increasing subsequences is interesting by itself and next we would like to give an algorithm with time complexity $O(n \log n)$ that performs the minimum increasing subsequence decomposition of $\tau \in \Sigma_n$.

**Algorithm 7.3.4.** FINDING MINIMUM NUMBER OF INCREASING SUBSEQUENCES

***Input***:

*A permutation $\tau = (\tau_1, \tau_2, \cdots, \tau_n)$ of the integer set $[n]$;*

***Output***:

*$m$ increasing subsequences $S_1, S_2, \ldots, S_m$;*

$m \leftarrow 1$, $S_1 \leftarrow \{\tau_1\}$, $T \leftarrow \{\tau_1\}$;

***for*** $i = 2$ ***to*** $n$ ***do***

  ***if*** *$\tau_i$ is smaller than all the integers in $T$* ***then***

    $m \leftarrow m + 1$, $S_m \leftarrow \{\tau_i\}$, $T \leftarrow T \cup \{\tau_i\}$;

  ***else***

    *Find the largest integer $p$ in $T$ that is less than $\tau_i$;*

    *Suppose $p$ is contained in $S_j$;*

    $S_j \leftarrow (S_j, \tau_i)$;

    *Replace $p$ by $\tau_i$ in $T$.*

***end for***

Note that, at each step, the set $T$ represents the last entries in the increasing subsequences considered up to that point.

**Theorem 7.3.5.** *The output of Algorithm* 7.3.4 *satisfies*

1. $m = m_2$;
2. *the subsequences $S_1, S_2, \cdots, S_m$ are an increasing subsequence decomposition of $\tau$.*

*The time complexity of Algorithm* 7.3.4 *is* $O(n \log n)$.

Figure 7.1 shows the lower and upper bounds on $t_1^*(\tau, \ell_0)$ averaged over all $n!$ permutations of $\tau \in \Sigma_n$. Also shown, in the curve labeled "No parallel," is the number of rounds required for "push-to-top" programming [48]. The parallel programming scheme requires roughly $\frac{1}{3}$ of the number of programming rounds compared to the push-to-top scheme.

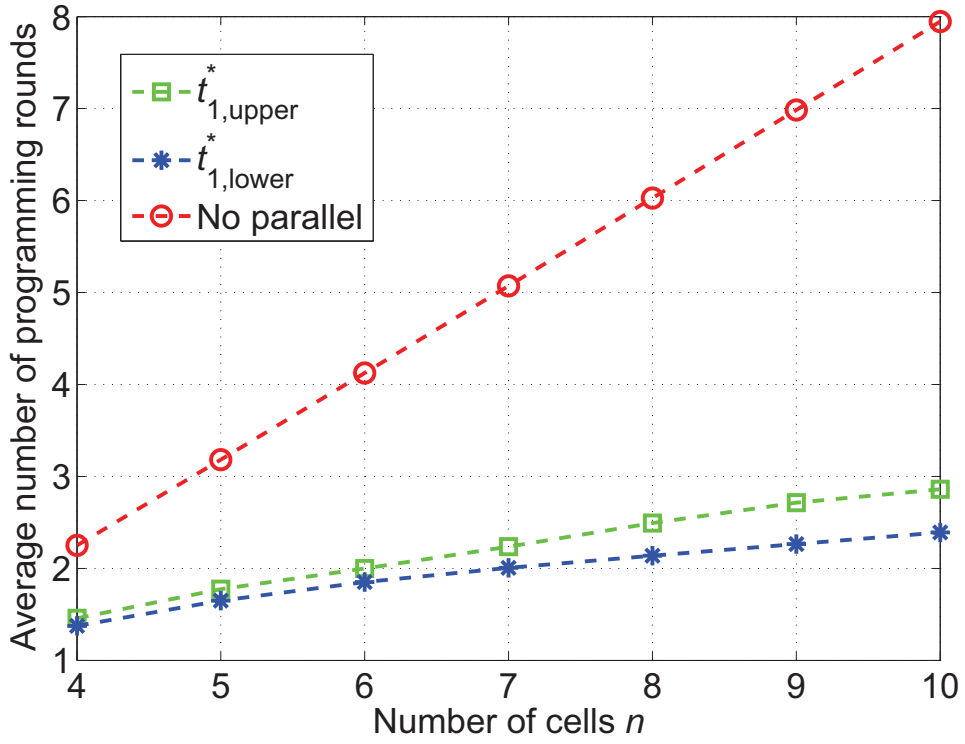


**Figure 7.1**: Lower and upper bounds on $t_1^*(\tau, \ell_0)$

For the case of small $n$, the optimal $t_1^*(\tau, \ell_0)$ can be derived by examining each $\tau \in \Sigma_n$. For example, if $n = 3$, then

$$t_1^*(\tau, \ell_0) = \begin{cases} 0, \text{ if } \tau = (1,2,3); \\ 2, \text{ if } \tau = (3,2,1); \\ 1, \text{ otherwise.} \end{cases}$$

Table 7.1 shows $t_1^*(\tau, \ell_0)$ for $n = 4$. In the table, $t_1^*((3142), \ell_0) = 1$ if $\ell_{4,0} - \ell_{3,0} \geqslant 2$ and $\ell_{2,0} - \ell_{1,0} \geqslant 2$; and $t_1^*((3142), \ell_0) = 2$, otherwise.

**Table 7.1**: $t_1^*(\tau, \boldsymbol{\ell}_0)$ as a function of $\tau$

| $\tau$ | $t^*$ | $\tau$ | $t^*$ | $\tau$ | $t^*$ | $\tau$ | $t^*$ |
|---|---|---|---|---|---|---|---|
| 1234 | 0 | 2134 | 1 | 3124 | 1 | 4123 | 1 |
| 1243 | 1 | 2143 | 1 | 3142 | - | 4132 | 2 |
| 1324 | 1 | 2314 | 1 | 3214 | 2 | 4213 | 2 |
| 1342 | 1 | 2341 | 1 | 3241 | 2 | 4231 | 2 |
| 1423 | 1 | 2413 | 1 | 3412 | 1 | 4312 | 2 |
| 1432 | 2 | 2431 | 2 | 3421 | 1 | 4321 | 2 |

**Conjecture 7.3.6** For each $\tau \in \Sigma_n$, there exists an $\boldsymbol{\ell}_0$, such that $t_1^*(\tau, \boldsymbol{\ell}_0)$ equals the lower bound $\lceil \log m_2 \rceil$; i.e., Theorem 7.3.2 provides a tight universal lower bounds on $t_1^*(\tau, \boldsymbol{\ell}_0)$.

## 7.4 Rank modulation maximizing the number of updates

As in the previous section, we assume $\alpha_i = 1, \forall i \in [n]$ and both $\ell_{i,j}$ and $V_j$ are integers, for $i \in [n], j \in [t]$. Suppose that the rank of the initial cell-state vector $\boldsymbol{\ell}_0$ is the identity permutation $\sigma = (1, \ldots, n)$ and let the target permutation be $\tau \in \Sigma_n$. According to [32], in order to maximize the number of updates (or equivalently minimize the maximum cell level after programming), the final cell-state vector satisfies $\ell_{\tau_1,t} = \ell_{\tau_1,0}$, and $\ell_{\tau_i,t} = \max\{\ell_{\tau_{i-1},t} + 1, \ell_{\tau_i,0}\}, i \in [2:n]$.

Our goal is to minimize the number of programming rounds $t$ such that the final cell-state vector is $\boldsymbol{\ell}_t$. Define the set of cell-level increments $\mathcal{I} = \{I_1, \ldots, I_m\}$ to be the set of distinct integers in $\boldsymbol{\ell}_t - \boldsymbol{\ell}_0$. Without loss of generality, we assume $0 < I_1 < I_2 < \cdots < I_m$. The cell programming problem can be formulated as follows: given $\mathcal{I}$, minimize $t$ such that there exists $\boldsymbol{V}^T = (V_1, \ldots, V_t) \in \mathbb{Z}_+^t$ and for each $I_i \in \mathcal{I}, i \in [m]$, there exists a $\boldsymbol{b}_i \in \{0,1\}^t$ such that $\boldsymbol{V}^T \cdot \boldsymbol{b}_i = I_i$. The pair $(\boldsymbol{V}, \mathbf{B})$ is called an **optimal pair** if $(\boldsymbol{V}, \mathbf{B})$ achieves the minimum number of rounds $t$. A vector $\boldsymbol{V}$ is called **optimal** if there exists a $\mathbf{B}$ such that $(\boldsymbol{V}, \mathbf{B})$ is an optimal pair.

For $\boldsymbol{V} \in \mathbb{Z}_+^t$, we call the set of integers $\{z \in \mathbb{Z} | z = \boldsymbol{V}^T \cdot \boldsymbol{b} \text{ for some } \boldsymbol{b} \in \{0,1\}^t\}$ the **span** of $\boldsymbol{V}$, denoted by $S(\boldsymbol{V})$. If $\mathcal{I} \subset S(\boldsymbol{V})$, we say $\boldsymbol{V}$ **covers** $\mathcal{I}$. Therefore, the cell programming problem translates to finding a vector $\boldsymbol{V}$ of minimum length that covers a given integer set.

**Example 7.4.1.** Suppose $\mathcal{I} = \{2, 5, 7, 8, 10\}$. Let $\boldsymbol{V} = (2, 3, 5)^T$, and the $\boldsymbol{b}_i$'s are chosen such that

$$(2,3,5) \begin{bmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \end{bmatrix} = (2,5,7,8,10).$$

Therefore, there exists a $\boldsymbol{V}$ of length 3 that covers $\mathcal{I}$.

This problem is related to a well-known NP-complete problem, Subset Sum Problem. Finding the optimal $\boldsymbol{V}$ for each given $\mathcal{I}$ is therefore a hard task, so we will settle for finding a solution which, though not necessarily optimal, compares favorably to the simple push-to-top [48] solution.

First we provide a general upper bounds on $t_2^*(\tau, \ell_0)$ in the following theorem.

**Theorem 7.4.1.** *The minimum possible number of programming rounds satisfies*

$$t_2^* \leqslant \min\{\lceil \log(I_m + 1) \rceil, 1 + \lceil \log(I_m - I_1 + 1) \rceil, m\}.$$

**Proof.** If $\boldsymbol{V}$ is chosen as $(2^0, 2^1, \ldots, 2^{\lceil \log(I_m+1) \rceil - 1})$, then it is guaranteed that $\boldsymbol{V}$ covers

$$[2^{\lceil \log(I_m+1) \rceil} - 1] \supset [I_m] \supset \mathcal{I}.$$

Therefore, $t_2^* \leqslant \lceil \log(I_m + 1) \rceil$.

If $\boldsymbol{V}$ is chosen as $(I_1, 2^0, 2^1, \ldots, 2^{\lceil \log(I_m - I_1 + 1) \rceil - 1})$, then it is guaranteed that $\boldsymbol{V}$ covers $[I_1 : I_1 + 2^{\lceil \log(I_m - I_1 + 1) \rceil} - 1] \supset [I_1 : I_m]$. Therefore, $t_2^* \leqslant 1 + \lceil \log(I_m - I_1 + 1) \rceil$.

If $\boldsymbol{V}$ is chosen as $(I_1, I_2, \ldots, I_m)$, then it is guaranteed that $\boldsymbol{V}$ covers $\mathcal{I}$. Therefore, $t_2^* \leqslant |\mathcal{I}| = m$. ■

Here are some further results and conjectures related to this parallel programming problem.

**Proposition 7.4.2.** *There exists an optimal $\boldsymbol{V}$ such that all the elements in $\boldsymbol{V}$ are distinct.*

**Proof.** The proof is omitted due to space limitations. ■

**Conjecture 7.4.3** If the integrality constraint on $\boldsymbol{V}$ is loosened, requiring only that $\boldsymbol{V} \in \mathbb{R}_+^t$, then there still exists an optimal $\boldsymbol{V}$ such that $\boldsymbol{V} \in \mathbb{Z}_+^t$; i.e., $t_2$ is achieved using integer-valued voltages $\boldsymbol{V}$.

Next we present an algorithm that searches for a short, but not necessarily minimum-length, vector $\boldsymbol{V}$ that covers the given integer set $\mathcal{I}$. Theorem 7.4.1 suggests that $I_m$, the largest elements in $\mathcal{I}$, and $|\mathcal{I}|$, the cardinality of $\mathcal{I}$ have a strong influence on the size and composition of an optimal $\boldsymbol{V}$. Note that once $V_1$ is fixed, we can reduce all elements greater than $V_1$ by $V_1$, and reformulate the problem into that of finding a vector $\tilde{\boldsymbol{V}}$ that covers $\mathcal{I}_{\text{new}} = \{i \in \mathcal{I} | i <$

$V_1\} \cup \{i|(i+V_1) \in \mathcal{I}\}$. The following algorithm is based on an iterative, greedy search for $V_1$ that, in each iteration, chooses $V_1$ to minimize $|\mathcal{I}_{\text{new}}|$ and, should more than one such $V_1$ exist, chooses one that minimizes the largest element in $\mathcal{I}_{\text{new}}$.

**Algorithm 7.4.4.** FINDING $\boldsymbol{V}$ THAT COVERS $\mathcal{I}$

***Input***:
*an integer set* $\mathcal{I} = \{I_1, \ldots, I_m\}$;
***Output***:
*an integer vector* $\boldsymbol{V}$ *that covers* $\mathcal{I}$;
$\boldsymbol{V} \leftarrow \emptyset$;
***while*** $\mathcal{I} \not\subset S(\boldsymbol{V})$
$\mathcal{I} \leftarrow$ *ascending_sort*$(\mathcal{I})$;
*min_size* $\leftarrow |\mathcal{I}|$;
*max_elem* $\leftarrow I_1$;
$V_1 \leftarrow 0$;
***for*** $v = 1$ ***to*** $I_m$ ***do***
$\mathcal{I}_{new} \leftarrow$ *reduced_set*$(\mathcal{I}, v)$;
***if*** $|\mathcal{I}_{new}| <$ *min_size* ***then***
$V_1 \leftarrow v$;
*min_size* $\leftarrow |\mathcal{I}_{new}|$;
*max_elem* $\leftarrow \max\{\mathcal{I}_{new}\}$;
***else***
***if*** $|\mathcal{I}_{new}| =$ *min_size* ***then***
$I_{new}^{\max} \leftarrow= \max\{\mathcal{I}_{new}\}$;
***if*** $I_{new}^{\max} <$ *max_elem* ***then***
$V_1 \leftarrow v$;
*max_elem* $\leftarrow \max\{\mathcal{I}_{new}\}$;
***end for***
$\boldsymbol{V} \leftarrow (\boldsymbol{V}, V_1)$; // *append* $V_1$ *to* $\boldsymbol{V}$
***end while***

In Algorithm 7.4.4, ascending_sort$(\mathcal{I})$ sorts $\mathcal{I}$ in ascending order and $\max\{\mathcal{I}_{\text{new}}\}$ returns the maximum element of a set $\mathcal{I}_{\text{new}}$. For a given $v \in [1 : I_m]$, reduced_set$(\mathcal{I}, v)$, calculated using Algorithm 3, generates $\mathcal{I}_{\text{new}}$ for the next iteration and minimizes $|\mathcal{I}_{\text{new}}|$ and $I_{\text{new}}^{\max}$.

**Algorithm 7.4.5.** $\mathcal{I}_{\text{NEW}}$=REDUCED_SET$(\mathcal{I}, v)$

***Input***:

*a sorted integer set* $\mathcal{I} = \{I_1, \ldots, I_m\}$ *in ascending order and an integer* $v \in [1 : I_m]$;

***Output***:

*an integer set* $\mathcal{I}_{new}$;

*flag* $\leftarrow$ *all-zero vector of length* $m$;

***for*** $i = m$ ***downto*** $1$ ***do***

  ***if*** $I_i > v$ ***and*** *flag*$[i] == 0$ ***then***

    $I_i \leftarrow I_i - v$;

    *flag*$[k] \leftarrow 1$ *for all* $k$ *such that* $I_k = I_i$;

***end for***

$\mathcal{I}_{new} \leftarrow$ *the set of distinct elements of* $\mathcal{I}$;

**Proposition 7.4.6.** *Suppose* $I_m < N$ *for some* $N \in \mathbb{Z}$. *Then the complexity of Algorithm* 7.4.5 *is* $O(N)$ *and that of Algorithm* 7.4.4 *is* $O(N^2 \log N)$.

Figure 7.2 shows results obtained using Algorithm 2 (and Algorithm 3). The horizontal axis represents the number of cells $n$ and the vertical axis represents the number of programming rounds $t$ averaged over all $n!$ target rank permutations $\tau \in \Sigma_n$. Also shown in the figure is the upper bound of Theorem 7.4.1, the number of rounds using the push-to-top strategy [32, 48], and the optimal number obtained using a brute-force, exhaustive search that minimizes $t$ for each $\tau \in \Sigma$. We note that the number of programming rounds obtained using Algorithm 7.4.4 (and 7.4.5) is only 2.5% above the optimal result when $n = 10$.

## 7.5 Conclusion

Rank modulation is a technique for representing stored information in an ordered set of flash memory cells by a permutation that reflects the ranking of their voltage levels. In this chapter, we consider two figures of merit that can be used to compare parallel programming algorithms for rank modulation. These two criteria represent different trade-offs between the programming speed and the lifetime of flash memory cells. In the first scenario, we want to find the minimum number of programming rounds required to increase a specified cell-level vector $\boldsymbol{\ell}_0$ to a cell-level vector corresponding to a target rank permutation $\tau$, with no restriction on the maximum allowable cell level. We derive lower and upper bounds on this number, denoted by $t_1^*(\tau, \boldsymbol{\ell}_0)$. In the second scenario, we seek an efficient programming strategy to achieve

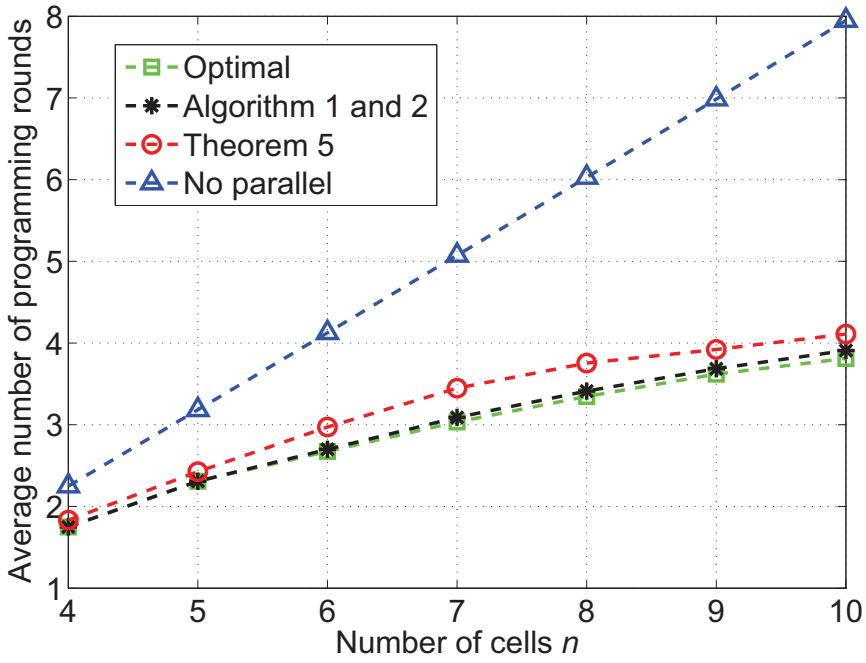


**Figure 7.2**: Number of programming rounds needed to achieve $\ell_t$

a cell-level vector $\ell(\tau)$ consistent with the target permutation $\tau$, such that the maximum cell level after programming is minimized. Equivalently, this strategy maximizes the number of information update cycles supported by the device before requiring a block erasure. We derive upper bounds on the minimum number of programming rounds required to achieve cell-level vector $\ell(\tau)$, denoted by $t_2^*(\tau, \ell_0)$, and propose a programming algorithm for which the resultant number of programming rounds is close to $t_2^*(\tau, \ell_0)$.

## Acknowledgments

This chapter is in part a reprint of the material in the paper: Minghai Qin, Anxiao Jiang, and Paul H. Siegel, "Parallel programming of rank modulation", in Proc. IEEE International Symposium of Information Theory (ISIT) 2013, Istanbul, Turkey, July 2013.

I would also like to thank Aman Bhatia and Bing Fan for helpful discussions.

## Bibliography


[1] L. V. Ahlfors, *Complex Analysis*. New York: McGraw-Hill, 1966.

[2] R. Ahlswede and Z. Zhang, "Coding for write-efficient memory," *Inform. and Comput.*, vol. 83, no. 1, pp. 80–97, October 1989.

[3] N. Alon and J. Spencer, *The Probabilistic Method*. New York: John Wiley Inc., 1992.

[4] T. Berger, *Rate Distortion Theory*. Englewood Cliffs, NJ: Prentice Hall, 1971.

[5] A. Berman and Y. Birk, "Constrained flash memory programming," in *Proc. IEEE Int. Symp. Inform. Theory*, St. Petersburg, Russia, July - August 2011, pp. 2128 – 2132.

[6] A. Bhatia, A. R. Iyengar, and P. H. Siegel, "Multilevel 2-cell $t$-write codes," in *Proc. IEEE Information Theory Workshop*, Lausanne, Switzerland, September 2012, p. 101 105.

[7] A. Bhatia, M. Qin, A. R. Iyengar, B. M. Kurkoski, and P. H. Siegel, "Lattice-based WOM codes for multilevel flash memories," *IEEE J. Selected Areas in Comm.*, vol. 32, no. 5, pp. 933 – 945, May 2014.

[8] V. Bohossian, A. Jiang, and J. Bruck, "Buffer codes for asymmetric multi-level memory," in *Proc. IEEE Int. Symp. Inform. Theory*, June 2007, pp. 1186–1190.

[9] A. Brouwer, J. Shearer, N. Sloane, and W. Smith, "A new table of constant weight codes," *IEEE Trans. Inform. Theory*, vol. 36, no. 6, pp. 1334–1380, November 1990.

[10] G. Burr, M. Breitwisch, M. Franceschini, D. Garetto, K. Gopalakrishnan, B. Jackson, B. Kurdi, C. Lam, L. Lastras-Montaño, A. Padilla, B. Rajendran, S. Raoux, and R. Shenoy, "Phase change memory technology," *Journal of Vacuum Science and Technology*, vol. 28, no. 2, pp. 223–262, March 2010.

[11] D. Callan, October 2012, Personal communication.

[12] P. Cappelletti, C. Golla, P. Olivo, and E. Zanoni, *Flash Memories*. Kluwer Academic Publishers, 1st Edition, 1999.

[13] Y. Cassuto and E. Yaakobi, "Practical re-write codes with access considerations," in *Non-Volatile Memories Workshop*, March 2012.

[14] A. Cauchy, *Analyse algébrique*, 1821.

[15] G. D. Cohen, "A nonconstructive upper bound on covering radius," *Discrete Mathematics*, vol. 29, no. 3, pp. 352–353, May 1993.

[16] G. D. Cohen and G. Zemor, "Write-isolated memories (WIMs)," *Discrete Math.*, vol. 114, no. 1-3, pp. 105–113, April 1993.

[17] J. H. Conway and N. J. A. Sloane, *Sphere Packings, Lattices and Groups*. New York, NY, Springer-Verlag, 3rd ed., 1999.

[18] T. Cover, "Enumerative source encoding," *IEEE Trans. Inform. Theory*, vol. 19, no. 1, pp. 73 – 77, January 1973.

[19] S. Datta and S. W. McLaughlin, "An enumerative method for runlength-limited codes: permutation codes," *IEEE Trans. Inform. Theory*, vol. 45, no. 6, pp. 2199–2204, September 1999.

[20] ——, "Optimal block codes for m-ary runlength-constrained channels," *IEEE Trans. Inform. Theory*, vol. 47, no. 5, pp. 2069–2078, July 2001.

[21] G. Dong, S. Li, and T. Zhang, "Using data postcompensation and predistortion to tolerate cell-to-cell interference in MLC NAND flash memory," *IEEE Transactions on Circuits and Systems*, vol. 57, no. 10, pp. 2718 – 2728, October 2010.

[22] A. El Gamal and Y.-H. Kim, *Network Information Theory*. Cambridge: Cambridge University Press, 2011.

[23] S. Forchhammer and T. V. Laursen, "A model for the two-dimensional no isolated bits constraint," in *Proc. IEEE Int. Symp. Inform. Theory*, Seattle, Washington, July 2006, pp. 1189 – 1193.

[24] G. D. Forney, R. G. Gallager, G. R. Lang, F. M. Longstaff, and S. U. H. Qureshi, "Efficient modulation for band-limited channels," *IEEE J. Selected Areas in Comm.*, vol. 2, no. 5, p. 632 647, September 1984.

[25] G. D. Forney and L.-F. Wei, "Multidimensional constellations part 1: Introduction, figures of merit, and generalized cross constellations," *IEEE J. Selected Areas in Comm.*, vol. 7, no. 6, p. 877 892, August 1989.

[26] F. Freitas and W. Wickle, "Storage-class memory: The next storage system technology," *IBM Journal of Research and Development*, vol. 52, no. 4/5, pp. 439–447, 2008.

[27] F. Fu, "Maximum information bits stored in reusable memory," *Chinese Science Bulletin*, vol. 40, no. 15, pp. 1241–1244, August 1995.

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

[29] F. Fu and R. Yeung, "On the capacity and error-correcting codes of write-efficient memories," *IEEE Trans. Inform. Theory*, vol. 46, no. 7, pp. 2299 – 2314, November 2000.

[30] R. Gabrys and L. Dolecek, "Characterizing capacity achieving write once memory codes for multilevel flash memoriess," in *Proc. IEEE Int. Symp. Inform. Theory*, July-August 2011, pp. 2517–2521.

[31] R. Gabrys, E. Yaakobi, L. Dolecek, S. Kayser, P. H. Siegel, A. Vardy, and J. K. Wolf, "Non-binary wom-codes for multilevel flash memories," in *Proc. IEEE Inform. Theory Workshop*, Paraty, Brazil, October 2011.

[32] E. Gad, A. Jiang, and J. Bruck, "Compressed encoding for rank modulation," in *Proc. IEEE Int. Symp. Inform. Theory*, St. Petersburg, Russia, July–August 2011, pp. 884–888.

[33] J. Hadamard, "Sur le rayon de convergence des séries ordonnées suivant les puissances d'une variable," *C. R. Acad. Sci. Paris 106*, pp. 259–262.

[34] S. Halevy, J. Chen, R. M. Roth, P. H. Siegel, and J. K. Wolf, "Improved bit-stuffing bounds on two-dimensional constraints," *IEEE Trans. Inform. Theory*, vol. 50, no. 5, pp. 824 – 838, May 2004.

[35] D. Haugland, "A bidirectional greedy heuristic for the subspace selection problem," *Lecture Notes in Computer Science*, vol. 4638, pp. 162–176, August 2007.

[36] C. Heegard, "On the capacity of permanent memory," *IEEE Trans. Inform. Theory*, vol. 31, no. 1, pp. 34–42, January 1985.

[37] C. Heegard and A. El Gamal, "On the capacity of computer memory with defects," *IEEE Trans. Inform. Theory*, vol. 29, no. 5, pp. 731 – 739, September 1983.

[38] P. Henry, "Zero disparity coding system," *U.S. Patent No. 4,309,694*, 1982.

[39] K. Immink and J. Weber, "Very efficient balanced codes," *IEEE Jounal on Selected Areas in Communications*, vol. 28, no. 2, pp. 188–192, February 2010.

[40] K. Immink, J. Weber, and H. Ferreira, "Balanced runlength limited codes using knuth's algorithm," in *Proc. IEEE Int. Symp. Inform. Theory*, August 2011, pp. 317–320.

[41] K. S. Immink, P. Siegel, and J. Wolf, "Codes for digital recorders," *IEEE Trans. Inform. Theory*, vol. 44, no. 6, pp. 2260–2299, Oct. 1998.

[42] A. Jiang, V. Bohossian, and J. Bruck, "Floating codes for joint information storage in write asymmetric memories," in *Proc. IEEE Int. Symp. Inform. Theory*, June 2007, pp. 1166–1170.

[43] ——, "Rewriting codes for joint information storage in flash memories," *IEEE Trans. Inform. Theory*, vol. 56, no. 10, pp. 5300–5313, October 2010.

[44] A. Jiang, J. Bruck, and H. Li, "Constrained codes for phase-change memories," in *Proc. IEEE Inform. Theory Workshop*, Dublin, Ireland, August-September 2010.

[45] A. Jiang, M. Langberg, M. Schwartz, and J. Bruck, "Universal rewriting in constrained memories," in *Proc. IEEE Int. Symp. Inform. Theory*, Seoul, Korea, July 2009, pp. 1219–1223.

[46] A. Jiang and H. Li, "Optimized cell programming for flash memories," in *Proc. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, Victoria, BC, Canada, August 2009, pp. 914–919.

[47] A. Jiang, H. Li, and J. Bruck, "On the capacity and programming of flash memories," *IEEE Trans. Inform. Theory*, vol. 58, no. 3, pp. 1549–1564, March 2011.

[48] A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck, "Rank modulation for flash memories," *IEEE Trans. Inform. Theory*, vol. 55, no. 6, pp. 2659 – 2673, June 2009.

[49] A. Jiang and Y. Wang, "Rank modulation with multiplicity," in *Proc. Globecom 2010*, Miami, FL, USA, December 2010, pp. 1866–1870.

[50] A. Kato and K. Zeger, "On the capacity of two-dimensional run-length constrained channels," *IEEE Trans. Inform. Theory*, vol. 45, no. 5, pp. 1527 – 1540, July 1999.

[51] S. Kayser, E. Yaakobi, P. H. Siegel, A. Vardy, and J. K. Wolf, "Multiple-write WOM-codes," in *Proc. 48-th Annual Allerton Conference on Communication, Control and Computing*, Monticello, IL, September 2010.

[52] D. E. Knuth, "Efficient balanced codes," *IEEE Trans. Inform. Theory*, vol. 32, no. 1, pp. 51 – 53, January 1986.

[53] B. Kurkoski, "Notes on a lattice-based wom construction that guarantees two writes," in *Proc. 34th Symp. Information Theory and Applications*, Ousyuku, Iwate, Japan, November - December 2011, pp. 520–524.

[54] ——, "Lattice-based wom codebooks that allow two writes," in *International Symposium on Information Theory and its Applications*, Honolulu, Hawaii, October 2012, p. 101 105.

[55] J. Lafferty and A. Vardy, "Ordered binary decision diagrams and minimal trellises," *IEEE Transactions on Computers*, vol. 48, no. 9, pp. 971 – 986, September 1999.

[56] L. Lastras-Montaño, M. Franceschini, T. Mittelholzer, J. Karidis, and M. Wegman, "On the lifetime of multilevel memories," in *Proc. IEEE Int. Symp. Inform. Theory*, Seoul, Korea, July 2009, pp. 1224–1228.

[57] J. Lee and V. K. Madisetti, "Constrained multitrack RLL codes for the storage channel," *IEEE Transaction on Magnetics*, vol. 31, no. 3, pp. 2355 – 2364, May 1995.

[58] J.-D. Lee, S.-H. Hur, and J.-D. Choi, "Effects of floating-gate interference on NAND flash memory cell operation," *IEEE Electron Device Letters*, vol. 23, no. 5, pp. 264 – 266, May 2002.

[59] Q. Li, "WOM codes against inter-cell interference in NAND memories," in *Proc. 49-th Annual Allerton Conference on Communication, Control and Computing*, Monticello, IL, September 2011, pp. 1416 – 1423.

[60] H. T. Lue, T. H. Hsu, S. Y. Wang, E. K. Lai, K. Y. Hsieh, R. Liu, and C. Y. Lu, "Study of incremental step pulse programming (ISPP) and STI edge effiect of BE-SONOS NAND flash," in *Proc. IEEE int. Symp. on Reliability Physiscs*, vol. 30, no. 11, May 2008, pp. 693–694.

[61] B. H. Marcus, R. M. Roth, and P. H. Siegel, *Constrained Systems and Coding for Recording Channels*. Handbook of Coding Theory (V. S. Pless and W. C. Huffman, eds.), ch. 20, Elsevier Science, 1998.

[62] R. Mascella and L. G. Tallini, "On symbol permutation invariant balanced codes," in *Proc. IEEE Int. Symp. Inform. Theory*, Adelaide, Australia, September 2005, pp. 4–9.

[63] ——, "Efficient m-ary balanced codes which are invariant under symbol permutation," *IEEE Trans. Computers*, vol. 55, no. 8, pp. 929–946, August 2006.

[64] J. Moon and B. Brickner, "Maximum transition run codes for data storage systems," *IEEE Trans. Magn.*, vol. 32, no. 5, pp. 3992–3994, Sep. 1996.

[65] Z. Nagy and K. Zeger, "Asympototic capacity of two-dimensional channels with checkerboard constraints," *IEEE Trans. Inform. Theory*, vol. 49, no. 9, pp. 2115–2125, September 2003.

[66] B. K. Natarajan, "Sparse approximate solutions to linear systems," *SIAM J. Comput.*, vol. 30, no. 2, pp. 227–234, April 1995.

[67] E. Ordentlich and R. M. Roth, "Two-dimensinal weight-constrained codes through enumeration bounds," *IEEE Trans. Inform. Theory*, vol. 46, no. 4, pp. 1292–1301, July 2000.

[68] A. Pirovano, A. Redaelli, F. Pellizzer, F. Ottogalli, M. Tosi, D. Ielmini, A. Lacaita, and R. Bez, "Reliability study of phase-change nonvolatile memories," *IEEE Trans. Device and Materials Reliability*, vol. 4, no. 3, pp. 422–427, September 2004.

[69] Q. Huang, S. Lin, and K. A. S. Abdel-Ghaffar, "Error-correcting codes for flash coding," *IEEE Trans. Inf. Theory*, vol. 57, no. 9, pp. 6097 – 6108, September 2011.

[70] M. Qin, E. Yaakobi, and P. H. Siegel, "Optimized cell programming for flash memories with quantizers," in *Proc. IEEE Int. Symp. Inform. Theory*, Cambridge, MA, USA, July 2012, pp. 1000–1004.

[71] R. Rivest and A. Shamir, "How to reuse a write-once memory," *Inform. and Contr.*, vol. 55, no. 1-3, pp. 1–19, December 1982.

[72] R. M. Roth, P. H. Siegel, and J. K. Wolf, "Efficient coding schemes for the hard-square model," *IEEE Trans. Inform. Theory*, vol. 47, no. 3, pp. 1166 – 1176, March 2001.

[73] R. Roth, July 2013, Personal Communication.

[74] C. Schensted, "Longest increasing and decreasing subsequences," *Canad. J. Math. 13*, pp. 179–191, 1961.

[75] C. E. Shannon, "A mathematical theory of communication," *Bell Systems Tech. J.*, vol. 27, p. 379  423, 1948.

[76] A. Shpilka, "Capacity achieving two-write wom codes," in *Latin American Symp. on Theoretical Informatics*, Arequipa, Peru, April 2012, pp. 631 – 642.

[77] N. J. A. Sloane, *The Online Encyclopedia of Integer Sequences*, https://oeis.org/.

[78] R. Stanley, "Increasing and decreasing subsequences and their variants," in *Proc. Internat. Cong. (Madrid 2006)*. American Mathematical Society, 2007, pp. 545–579.

[79] G. Stefano, S. Krause, M. Lübbeck, and U. Zimmermann, "On minimum *k*-modal partitions of permutations," *J. Discrete Alg. 6*, pp. 381–392, 2008.

[80] K.-D. Suh, B.-H. Suh, Y.-H. Lim, Y.-J. C. J-K. Kim, Y.-N. Koh, S.-S. Lee, S.-C. Kwon, B.-S. Choi, J.-S. Yum, J.-H. Choi, J.-R. Kim, and H.-K. Lim, "A 3.3V 32 Mb NAND flash memory with incremental step pulse programming scheme," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 11, pp. 1149–1156, November 1995.

[81] Y. Sun, “The statistic ”number of udu’s” in Dyck paths,” *Discrete Mathematics, 287*, pp. 177 – 186, July 2004.

[82] R. E. Swanson and J. K. Wolf, “A new class of two-dimensional RLL recording codes,” *IEEE Transations on Magnetics*, vol. 28, no. 6, pp. 3407 – 3416, November 1992.

[83] T. G. Swart and J. H. Weber, “Efficient balancing of q-ary sequences with parallel decoding,” in *Proc. IEEE Int. Symp. Inform. Theory*, Seoul, Korea, June - July 2009, pp. 1564–1568.

[84] I. Tal and R. M. Roth, “Convex programming upper bounds on the capacity of 2-D constraints,” *IEEE Trans. Inform. Theory*, vol. 57, no. 1, pp. 381–391, January 2011.

[85] L. G. Tallini and B. Bose, “Balanced codes with parallel encoding and decoding,” *IEEE Trans. Computers*, vol. 48, no. 8, pp. 794–814, August 1999.

[86] L. G. Tallini, R. M. Capocelli, and B. Bose, “Design of some new efficient balanced codes,” *IEEE Trans. Inform. Theory*, vol. 42, no. 3, pp. 790–802, May 1996.

[87] L. Wang, M. Qin, E. Yaakobi, Y.-H. Kim, and P. H. Siegel, “WOM with retained messages,” in *Proc. IEEE Int. Symp. Inform. Theory*, Cambridge, MA, USA, July 2012, pp. 1396–1400.

[88] L. Wang and Y.-H. Kim, “Sum-capacity of multiple-write noisy memory,” in *Proc. IEEE Int. Symp. Inform. Theory*, St. Petersburg, Russia, July–August 2011, pp. 2494–2498.

[89] J. Weber and K. Immink, “Knuth’s balanced codes revisited,” *IEEE Trans. Inform. Theory*, vol. 56, no. 4, pp. 1673–1679, April 2010.

[90] W. Weeks and R. Blahut, “The capacity and coding gain of certain checkerboard codes,” *IEEE Trans. Inform. Theory*, vol. 44, no. 3, pp. 1193–1203, May 1998.

[91] J. K. Wolf, A. D. Wyner, J. Ziv, and J. Korner, “Coding for a write-once memory,” *AT&T Bell Labs. Tech. J.*, vol. 63, no. 6, pp. 1089–1112, 1984.

[92] Y. Wu, “Low complexity codes for writing write-once memory twice,” in *Proc. IEEE Int. Symp. Inform. Theory*, Austin, Texas, June 2010, pp. 1928–1932.

[93] Y. Wu and A. Jiang, “Position modulation code for rewriting write-once memories,” *IEEE Trans. Inform. Theory*, vol. 57, no. 6, pp. 3692–3697, June 2011.

[94] E. Yaakobi, A. Jiang, P. H. Siegel, A. Vardy, and J. K. Wolf, “On the parallel programming of flash memory cells,” in *Proc. IEEE Inform. Theory Workshop*, Dublin, Ireland, August–September 2010.

[95] E. Yaakobi, S. Kayser, P. H. Siegel, A. Vardy, and J. K. Wolf, “Efficient two-write WOM-codes,” in *Proc. IEEE Inform. Theory Workshop*, Dublin, Ireland, August 2010.

[96] E. Zehavi and J. K. Wolf, “On runlength codes,” *IEEE Trans. Inform. Theory*, vol. 34, no. 1, pp. 45–54, January 1988.

[97] H. Zhou, A. Jiang, and J. Bruck, “Error-correcting schemes with dynamic thresholds in nonvolatile memories,” in *Proc. IEEE Int. Symp. Inform. Theory*, St. Petersburg, Russia, July–August 2011, pp. 2143 – 2147.