ALGORITHMS AND DATA REPRESENTATIONS FOR EMERGING NON-VOLATILE MEMORIES

A Dissertation

by

YUE LI

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2014

Major Subject: Computer Science

ABSTRACT

The evolution of data storage technologies has been extraordinary. Hard disk drives that fit in current personal computers have the capacity that requires tons of transistors to achieve in 1970s. Today, we are at the beginning of the era of non-volatile memory (NVM). NVMs provide excellent performance such as random access, high I/O speed, low power consumption, and so on. The storage density of NVMs keeps increasing following Moore's law. However, higher storage density also brings significant data reliability issues. When chip geometries scale down, memory cells (e.g. transistors) are aligned much closer to each other, and noise in the devices will become no longer negligible. Consequently, data will be more prone to errors and devices will have much shorter longevity.

This dissertation focuses on mitigating the reliability and the endurance issues for two major NVMs, namely, NAND flash memory and phase-change memory (PCM). Our main research tools include a set of coding techniques for the communication channels implied by flash memory and PCM. To approach the problems, at bit level we design error correcting codes tailored for the asymmetric errors in flash and PCM, we propose joint coding scheme for endurance and reliability, error scrubbing methods for controlling storage channel quality, and study codes that are inherently resisting to typical errors in flash and PCM; at higher levels, we are interested in analyzing the structures and the meanings of the stored data, and propose methods that pass such metadata to help further improve the coding performance at bit level. The highlights of this dissertation include the first set of write-once memory code constructions which correct a significant number of errors, a practical framework which corrects errors utilizing the redundancies in texts, the first report of the performance of polar codes for flash memories, and the emulation of rank modulation codes in NAND flash chips.

To my parents and my wife Xiheng.

ACKNOWLEDGEMENTS

I would like to thank my wonderful Ph. D. advisor Prof. Anxiao (Andrew) Jiang. I am truly grateful that Andrew picked me up three years ago. It is my honor to work with him. Andrew always gives me strong support, and he is always very patient. Without his constant guidance, I will never be able to accomplish any of the work in this dissertation. Andrew not only taught me how to be a hard-working researcher, but also taught me how to be a nice and understanding person in life. As I always say to my friends, being able to become one of Andrew's students is like wining a big lottery.

I would like to thank my another wonderful advisor Prof. Jehoshua (Shuki) Bruck for his endless support, and for having me in his group. Shuki is my academic grandfather. He always smiles as my grandfather did, and he indeed treats me as a family member. Shuki and Andrew are the professors that one always dreams to work with. I thank Shuki for showing me lots of interesting problems, and giving me lots of freedom to explore new directions. Shuki taught me how to be visionary in research, and how to be a nice person in life.

I would like to thank Prof. Eitan Yaakobi, Prof. Michael Langberg, and Prof. Jöerg Kliewer for being friends, teachers and collaborators. Discussions with them are always enjoyable and fruitful.

I would like to thank Prof. Andreas Klappenecker, Prof. Eun Jung Kim, and Prof. Krishna R. Narayanan for serving on my committee and providing precious feedback on my research despite their very busy schedules.

I would like to thank my ex-advisor Dr. Gabriel Dos Reis for making me a practical researcher. I would like to thank Prof. Jaakko Järvi for his help, understanding and suggestions.

I am grateful to my undergraduate advisor Prof. Yu Liu for her support. She taught me to be a positive and responsible person, and made me understand the significance of having good attitudes on anything that I work on. I would like to thank her for coming to visit me during my Ph. D. study.

I spent a wonderful summer at LSI. I'm grateful to Dr. Erich F. Haratsch and Dr. Hakim Alhussien, without whom I could never finish my project and publish the interesting results. I would like to thank Dr. Sundararajan Sankaranarayanan, Dr. Yu Cai, Dr. Jacob Chen, and Dr. Yingquan Wu for their suggestions on my research and career. I would like to thank Meysam Asadi and Dongyi Ye for accompanying me and going lunch with me during those joyful days.

I had a great working experience at the Department of Mathematics of Texas A&M. I would like to thank Prof. Ronald DeVore, Prof. Guergana Petrova for continuously supporting me. I'm grateful to Matthew Hielsberg for being friend, mentoring me , and teaching me how to build high quality software. I would like to thank Kristy Delvo-Vela for making the working environment extremely comfortable. I would like to thank Casey Rodriguez for working together with me.

I would like to thank Eyal En Gad for being a good friend, a nice academic uncle, a great research collaborator, and my patient Hebrew teacher. I would like to thank Dr. Farzad Farnoud (Hassanzadeh) for accompanying me in the lab when we worked overnight for deadlines. Farzad and I had lots of funny conversations and research discussions that I will never forget. I also would like to thank my labmates at Texas A&M Dr. Yue Wang, Qing Li, Xiang Chen, and Rui Niu. Working with them gives me lots of motivations for my research.

I would like to thank my close friends Youyou Wang, Hao Wang, Qiong Zhao, Lei Wang, Shi Pu, Xiabing Xu, Xiaojian Wu, Sheng Qiu, Xiaoyong Li, Wen Yuan, Yan Jing, Wen Li, Lin Sun and Tengteng Zhang for accompanying me at A&M. Those poker nights, dinner parties and trips with you are the most precious memory that makes me smile when I think of them. I would like to thank my former labmates Dr. Jacob N. Smith, Dr. Yuriy Solodkyy, Dr. Xiaolong Tang, Cindy (Hsin-Yi) Yeh and Wonseok Kim for being close friends and always supporting me.

Finally, the ultimate thanks go to my family. I'm grateful to my sweet and lovely wife Xiheng. Her beautiful smile, endless support, unconditional love and understanding are the forces that keep my research moving forward. I would like to give my deepest gratitude to my parents and my grandparents for raising me up and teaching me how to be a good person in life. Their strong encouragements, and endless love make me be brave to accept the challenges in my life and career. To them this dissertation is dedicated to.

TABLE OF CONTENTS

Page

ABSTRACT ii

DEDICATION iii

ACKNOWLEDGEMENTS iv

TABLE OF CONTENTS vii

LIST OF FIGURES x

LIST OF TABLES xiii

1. INTRODUCTION 1

    1.1 Basic Concepts of Flash Memory and PCM 2
    1.2 Data Reliability Challenges for Flash Memory and PCM 8
    1.3 Overview of Research Directions 10

2. BACKGROUND ON CODING FOR NONVOLATILE MEMORIES 14

    2.1 WOM Codes 14
    2.2 Error Correction Codes for Flash Memory and PCM 15
    2.3 Rank Modulation Codes 16

3. BIT-FIXING CODES FOR CORRECTING ASYMMETRIC ERRORS 17

    3.1 Bit-fixing Coding Scheme 18
        3.1.1 Numeral Systems for Cell Levels and Errors 18
        3.1.2 Bit-fixing Coding Scheme for Binary Representation 19
        3.1.3 Bit-fixing Coding Scheme for General Numeral Systems 24
        3.1.4 Achievable Rate of Bit-fixing Coding Scheme 25
    3.2 Optimal Labeling of Cell-levels 26
    3.3 Coding for Any Number of Cell Levels 30
    3.4 Performance Evaluation 30

4. CONTENT-ASSISTED ERROR CORRECTION FOR FILE RECOVERY . . . 33

    4.1 The Models of File Decoding 34
    4.2 The Content-Assisted Decoding Algorithms 37
        4.2.1 Creating Dictionaries 37
        4.2.2 Codeword Segmentation 37
        4.2.3 Ambiguity Resolution 41

4.2.4 Post-processing 45
4.3 Experiments 45
4.3.1 Implementation Detail 45
4.3.2 Evaluation 46

5. WOM CODES THAT CORRECT SYMMETRIC ERRORS 49

5.1 Basic Model 49
5.1.1 The Model for Rewriting 50
5.1.2 Polar Codes 51
5.2 Code Construction 52
5.2.1 Basic Code Construction with a Nested Structure 52
5.2.2 Extended Code Construction 56
5.3 Code Analysis for BSC 56
5.3.1 Correctness of the Code 56
5.3.2 The Size of $F_{\mathrm{WOM}(\alpha,\epsilon)} \cap F_{\mathrm{BSC}(p)}$ 57
5.3.3 Lower Bound to Sum-rate 59
5.4 Extensions 61
5.5 Experimental Results 62
5.5.1 Finding BSCs Satisfying $F_{\mathrm{BSC}(p)} \subseteq F_{\mathrm{WOM}(\alpha,\epsilon)}$ 63
5.5.2 Achievable Sum-rates for Nested Code 64
5.5.3 Achievable Sum-rates for General Code 64

6. ERROR CORRECTING TRAJECTORY CODES FOR PARTIAL INFORMATION REWRITING 66

6.1 Trajectory Codes 67
6.2 Error Correcting Trajectory Codes 68
6.2.1 Error Model and WOM Parameters 68
6.2.2 Code Construction 69
6.2.3 Analysis of the Correctness of the Construction 70
6.2.4 Code Analysis 71
6.3 A More Generalized Coding Scheme 74
6.3.1 Code Construction 74
6.3.2 Analysis of the Correctness of the Code 76
6.3.3 Code Analysis 77

7. WOM CODES THAT CORRECT WRITE ERRORS 80

7.1 WOM Channel Model 80
7.2 Polar Coding Notation 82
7.3 Coding Scheme 83
7.4 Optimality of the Code Rate 88
7.4.1 Fraction of Additional Cells is Negligible 88
7.4.2 The Rate of the Main Block 93
7.5 Probability of Decoding Error 96
7.6 Discussion 102

8. POLAR CHANNEL CODES IN MULTI-LEVEL FLASH MEMORIES 103

8.1 The Models of Flash Memories 105
8.1.1 Reading MLC 105
8.1.2 A Cascaded Noise Channel Model 106
8.2 Polar Codes for Flash Memories 108
8.2.1 Shortened Non-systematic Polar Codes 108
8.2.2 Shortened Systematic Polar Codes 110
8.2.3 Polar Codes with Bit-reversal Permutation 112
8.2.4 Performance Evaluation 113
8.3 Adaptive Decoding for Flash Memories 115

9. MULTI-PHASE SCRUBBING FOR PHASE-CHANGE MEMORIES 120

9.1 Multi-phase Memory Scrubbing 122
9.2 Code Rate Analysis 125
9.2.1 Channel Model 126
9.2.2 Code Rates 127
9.3 Decoding Error Rate Analysis 130
9.3.1 Decoding Error Rates without Metadata 130
9.3.2 Decoding Error Rates with Metadata 131
9.4 In-memory Error Detection 132
9.5 Performance Evaluation 133
9.5.1 Code Rates 133
9.5.2 Estimated Decoding Bit Error Rates 134
9.5.3 Simulation Results 135

10. EMULATING RANK MODULATION CODES IN FLASH MEMORIES . . . 137

10.1 Experimental Platform 137
10.1.1 Flash Test Board 137
10.1.2 NAND Flash Chips 138
10.1.3 Support from Flash Chip Manufacturers 139
10.2 Emulating Rank Modulation 139
10.3 Preliminary Results 142

11. SUMMARY AND DIRECTIONS OF FUTURE WORK 146

REFERENCES 149

LIST OF FIGURES

FIGURE Page

1.1 A floating-gate transistor. 3

1.2 Examples of SLC, MLC and TLC. 3

1.3 The threshold voltage distributions of multi-level cells. 4

1.4 The changes on programming currents during ISPP for different target cell levels [5]. 5

1.5 The process of two-step programming in multi-level cells. 6

1.6 An example of soft sensing with $k = 3$. 7

1.7 Different heating methods for operating PCM [8]. 8

3.1 Decoding process of bit-fixing scheme. Here thick solid arrows show how errors change cell levels to noisy levels. And thin dotted arrows show how decoding changes the noisy levels back to the original levels. For $i = 0, 1, 2$, the thin dotted arrows labeled by i correspond to the decoding of $\mathcal{C}_i$. 22

3.2 Labeling physical cell states with levels. (a) A straightforward labeling, where every physical state $s \in \{0, 1, \cdots, 15\}$ is labeled by level $\ell = s$. (b) A Gray-code labeling. (c) A new labeling. 28

3.3 Comparison of achievable rates (number of stored bits per cell). Here $q = 16$, $p = 0.01$, and L^+ increases from 1 to 6. Above: asymmetric errors, where $L^- = 0$. Below: bidirectional errors, where $L^- = 3$. 32

4.1 An example on correcting errors in the codeword of a text. 33

4.2 The channel model for data storage. 35

4.3 The work-flow of a channel decoder with content-assisted decoding. . . . 36

4.4 The examples of codeword segmentation. In Figure (c): A number in the table denotes the index of the first bit of an estimated space; a set of word means the subcodeword can be flipped to any of the word in the set. The cross $\times$ means a subcodeword can neither be flipped to represent a word nor to a text with at least two words. 41

4.5 Example of the mapping to trellis decoding. The word sets $\mathbf{W}_1 = \{\mathbf{w}_{1,1}, \mathbf{w}_{1,2}\}$, $\mathbf{W}_2 = \{\mathbf{w}_{2,1}, \mathbf{w}_{2,2}, \mathbf{w}_{3,2}\}$, $\mathbf{W}_3 = \{\mathbf{w}_{3,1}, \mathbf{w}_{3,2}, \mathbf{w}_{3,3}\}$ and $\mathbf{W}_4 = \{\mathbf{w}_{4,1}, \mathbf{w}_{4,2}\}$ respectively corresponds to the subcodewords $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$ and $\mathbf{x}_4$. 44

5.1 The WOM channel $WOM(\alpha, \epsilon)$. 53

5.2 (a) Nested code for $F_{\text{BSC}(p)} \subseteq F_{\text{WOM}(\alpha,\epsilon)}$. (b) General code. 53

5.3 Degrading the channel $WOM(\alpha, \epsilon^*)$ to $BSC(\alpha\epsilon^*)$. The two channels on the left and on the right are equivalent. 58

5.4 Degrading channel $WOM(\alpha, \frac{p}{\alpha})$ to $WOM(\alpha, \epsilon)$. Here $z = \frac{\alpha\epsilon - p}{\alpha - 2p}$. The two channels on the left and on the right are equivalent. 58

5.5 The frozen sets for channels $BSC(p)$, $WOM(\alpha, \epsilon)$, $WOM(\alpha, \frac{p}{\alpha})$ and $BSC(\alpha\epsilon)$. Here $p \leqslant \alpha\epsilon$. 59

5.6 Lower bound to achievable sum-rates for different error probability p. . . 62

5.7 Lower bound to achievable sum-rates for different error probability p. Here each rewriting step writes the same number of bits. 63

5.8 The maximum value of p found for which $F_{\text{BSC}(p)} \subseteq F_{\text{WOM}(\alpha,\epsilon)}$. 64

5.9 Sum-rates for different t obtained in experimental search using code length $N = 8192$, when $F_{\text{BSC}(p)} \subseteq F_{\text{WOM}(\alpha,\epsilon)}$. 65

6.1 The lower and upper bounds (marked by LB and UB) on the achievable code rates for different t and D. Here $\log_2 L = 2^{13}$, $C = 8$ and $p = 10^{-3}$. 74

6.2 The bounds to the achievable rates of the two constructions on different t and C. Here $\log_2 L = 2^{13}$, $p = 10^{-3}$, and $\epsilon_{i,j} = \frac{1}{2+t-j}$. 79

7.1 A binary WOM with write errors. 81

7.2 Different polarizations of $U_1^n = X_1^n G_n$. 84

8.1 An example of soft sensing with $k = 3$. 105

8.2 The cascaded noise channel of MLCs. 107

8.3 The performance of polar codes and LDPC codes at different PECs. . . . 114

8.4 The performance of polar codes with (a) different block lengths as well as (b) realistic and genie DSPs. 116

8.5 The theoretical decoding error rates of each subchannel at different PECs. Each curve denotes one subchannel. 118

8.6 The performance of polar codes constructed at fixed PECs throughout the lifetime of the flash chips. Soft sensing is used. 119

9.1 Resistance drift within 4-level cells with fixed thresholds for levels. Horizontal axis: the logarithmic resistance. Solid curves: the resistance distributions of the cell levels. Dotted vertical lines: the resistance thresholds. . 120

9.2 An example for illustrating multi-phase scrubbing. 125

9.3 The code rates of the three scrubbing schemes given different s. 134

9.4 The decoding BERs of each scheme with different scrubbing periods. . . 135

9.5 The BERs before every decoding during retention given $t_m = 3\,\mathrm{h}$. 136

10.1 The flash test board. 138

10.2 The first 4096 cells of a page in a Micron 34nm SLC flash chip with PP duration = 27 microseconds 140

10.3 The first 4096 cells of a page in a Micron 34nm SLC flash chip with PP duration = 27 microseconds 141

10.4 The average read error probabilities at different PECs without cell-to-cell interference. 144

10.5 The average read error probabilities at different PECs with cell-to-cell interference. 145

LIST OF TABLES

TABLE Page

4.1 The decoding BERs when the dictionaries are complete 46

4.2 The decoding BERs when the dictionaries are incomplete. 47

6.1 An example of the first construction. 71

6.2 An example of the second construction. 76

9.1 The configurations of PCM cells used for evaluations. 133

10.1 The NAND flash chips used in our experiments. 138

10.2 Error patterns of the two emulation schemes. 143

1. INTRODUCTION

Data storage technologies have been so widely integrated into our daily life that sometimes we do not even realize their existences. They are operating in small mobile electronic devices such as cameras, cell phones and tablets, they also live inside large scale high performance computing facilities such as data centers made of thousands of server racks. The evolution of storage technologies has been extraordinary. Hard disk drives (HDDs) in current personal computers can store data that requires tons of transistors to store forty years ago. Today, we are at the beginning of another evolution of data storage technologies—the era of nonvolatile memory (NVM). NVMs such as flash memories become ubiquitous because of their excellent performance. Compared to HDD, solid state drives (SSDs) made of flash memories are implemented purely with circuits, and provide much higher I/O speed, random access and less power consumption.

The density of NVMs keeps increasing following Moore's law. Although the area cost of NVMs has been significantly reduced, the much smaller chip geometry poses significant reliability challenges to NVMs. The main reason is that the basic storage units in NVMs (e.g. transistors in flash memories) are aligned much closer to each other, and more quantization levels are put into each unit to store more data, which greatly amplify the effects of different noise in the circuits. In fact, a recent study [24] has shown that the reliability of flash memories decreases exponentially with the storage density. Therefore, one of the ultimate goals for the data storage community is to study different methods for improving the reliability of NVMs while still preserving their excellent performance such as high storage density and speed. This dissertation reports part of our efforts towards reaching this goal. We will mainly focus on mitigating the reliability issues for two kinds of major emerging NVMs, namely, flash memory and phase-change memory (PCM). We briefly introduce these two kinds of memories in the next section.

1.1 Basic Concepts of Flash Memory and PCM

Flash Memories was first invented by Fujio Masuoka while he was working at Toshiba in 1980s. Very recently, flash has become one of the major storage media for both personal and enterprise computing. There are two kinds of flash memories, namely, NOR flash and NAND flash. Both of NOR and NAND flash memories use floating-gate transistor (which will be introduced shortly) as the basic storage unit. The major difference is that NOR flash provides random access capability at byte and word level, while NAND flash provides random access only at page level. Therefore, NOR flash is more suitable for code storage and execution and has wide applications in microprocessors. NAND flash is more suitable for data storage devices such as mobile devices and solid-state drives. As NOR flash provides random access at a finer level, compared to NAND flash, NOR flash chips have larger dimensions, and have lower density and higher price. In this dissertation, we will be focusing on NAND flash memories. However, the techniques studied here can be adapted to work for NOR flash due to the similarities between the two kinds of flash memories.

A flash memory chip is hierarchically organized following the order: plane, block, page, byte, and cells. For example, a NAND flash chip may have 2 planes, each plane has 2^{11} blocks, a block contains 256 pages where each page has 4320 bytes, and a byte contains 8 cells. A cell is the basic storage unit of flash memory, and it is implemented using floating-gate transistor (FGT) shown in Figure 1.1. A FGT has a control gate (CG), a channel consisting of a source and a drain, and a floating gate (FG) between the control gate and the channel. Cells are programmed to store data by injecting charge from the channel into the FG. To do so, a high voltage is applied on the control gate, and the electrons moving from source to drain will be pulled into the FG using the Fowler-

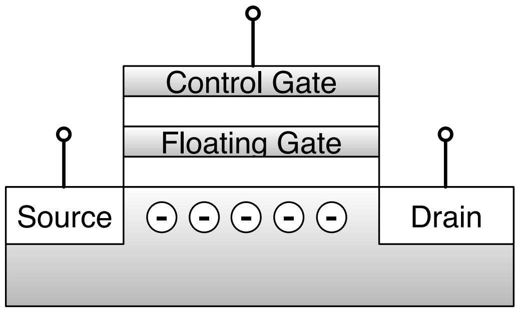


Figure 1.1: A floating-gate transistor.

Nordheim tunneling. The amount of charge in a floating gate transistor is quantized into multiple discrete levels to denote different cell states. In general, a cell that is quantized into q levels stores $\log_2 q$ bits. Currently, there are three main types of cells that have been used in commercial flash products, namely, single-level cell (SLC), multi-level cell (MLC), and triple-level cell (TLC) with densities 1, 2, and 3 bits per cell, respectively. Figure 1.2 shows the examples of the three kinds of cells with the cell levels being labeled using gray mapping (i.e. the bits representing two adjacent cell levels are different by only 1 bit). To read a cell, a small voltage is applied on the CG, and the charge amount

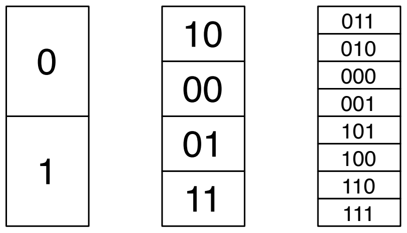


Figure 1.2: Examples of SLC, MLC and TLC.

of the FG is measured and compared with predetermined reference threshold voltages to determine the discrete cell level. Both programming and reading can be performed efficiently. To erase a cell, a high voltage in the opposite direction is applied on the control gate, which pulls the electrons out of the FG. This process causes significant interference on the neighboring cells, therefore erasing a cell currently requires erasing the whole block of cells, and the valid data stored in the other cells of the block to be erased need to be written into other blocks. The endurance of a cell is characterized by the number of times that the cell can be erased until it starts malfunction, e.g. a cell may get stuck and can no longer be programmed. Therefore, block erasures not only introduces significant latency, but also degrade the performance of memory.

We now explain the concepts of cell level distributions, page programming and reading. We will use MLCs for illustration purposes, and the cases for SLC and TLC is either an instance or a generalization of those of MLC. The threshold voltages of the cells in a flash memory block follow certain probability distributions. It is shown that such cell level distributions can be well approximated with Gaussian distribution [12]. Figure 1.3 shows an example of the distributions for MLC NAND flash memories. Cell levels are labeled using 2-bit Gray codes. In the following, we refer the left bit as the most significant bit (MSB) and the right bit as the least significant bit (LSB).

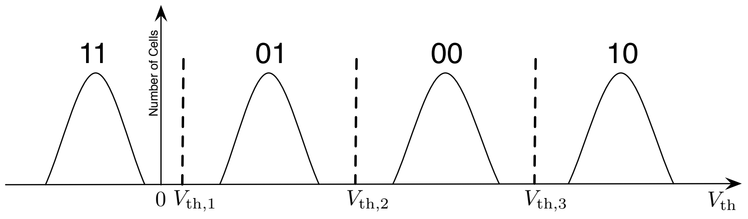


Figure 1.3: The threshold voltage distributions of multi-level cells.

Incremental step pulse programming (ISPP) is used for programming the cells in a page to the desired level. In this process, a small programming pulse is sent to each cell in each step, injecting a small amount of charge into the cells. Cells are programmed in parallel, and multiple programming steps are needed to reach the target threshold voltage of each cell. After each step, the cells are read and their current threshold voltages are compared with the target threshold voltages. The ISPP stops applying pulses to a cell if the cell has reached target threshold voltage, and the whole process terminates when each cell in the page reaches its target threshold voltage. Figure 1.4 shows the examples on how the currents change during ISPP. The amount of charge injected into a cell is adjusted in each step, and only a very small amount of charge gets injected into the cell in the last few steps.

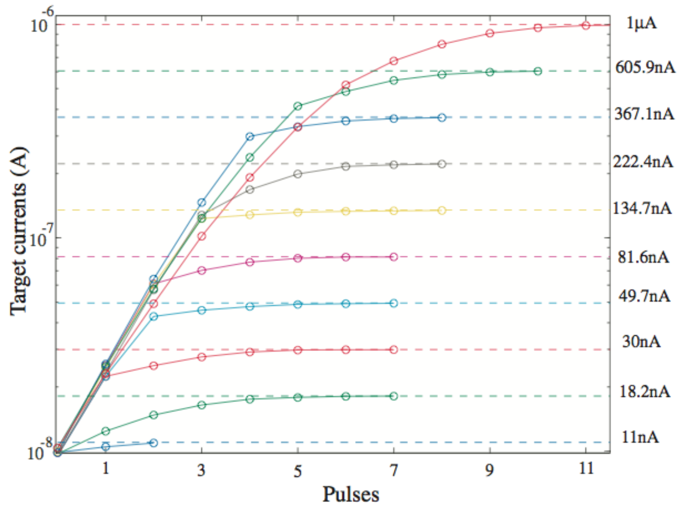


Figure 1.4: The changes on programming currents during ISPP for different target cell levels [5].

To store two bits in a MLC, a two-step programming method as shown in Figure 1.5 is used where one bit is stored in each step. Let the cell to be programmed be erased (Figure 1.5(a)). The first step writes LSB (Figure 1.5(b)). If the LSB to be written is 0, the cell level is raised to the intermediate state $x0$, otherwise the cell stays. The second step stores

MSB (Figure 1.5(c)). The LSB stored in the cell is first read using the reference threshold voltage between the states 11 and $x0$. If the cell is at state $x0$ and the MSB is 1, the cell is further raised to level 4, if the MSB is 0, the cell will be programmed to level 3. Similarly, if the current cell state is 11, the cell stays or be raised to level 2 according to the value of MSB.

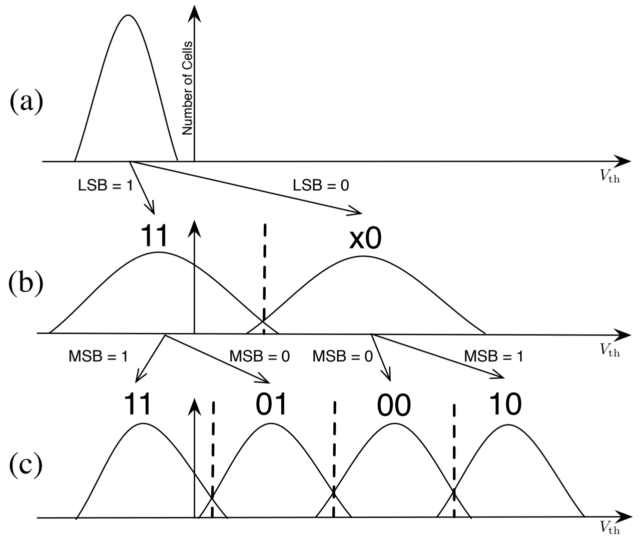


Figure 1.5: The process of two-step programming in multi-level cells.

Data are read either through hard or soft sensing. In each approach, MSB and LSB are read independently. Hard sensing returns the (possibly noisy) value of the bit being read, while soft sensing outputs the log-likelihood ratio (LLR) of the bit. Specifically, hard sensing uses one reference threshold voltage between two adjacent distributions as shown in Figure 1.3. Let the reference threshold voltages be $V_{\text{th},1}$, $V_{\text{th},2}$, and $V_{\text{th},3}$. To read LSB, the cell threshold voltage V_{th} is compared with $V_{\text{th},2}$, returning 0 if $V_{\text{th}} > V_{\text{th},2}$, and 1 otherwise. To read MSB utilizes both $V_{\text{th},1}$ and $V_{\text{th},3}$: when $V_{\text{th},1} < V_{\text{th}} < V_{\text{th},3}$, output 0, otherwise output 1. Soft sensing use k reference threshold voltages between two adjacent distributions [72]. Figure 1.6 shows an example for $k = 3$. For $i \in \{1, 2, 3\}$, $j \in \{1, 2, \cdots, k\}$, let $V_{\text{th},i,j}$ be the j-th reference threshold voltage of the k reference threshold voltages between level i and $i + 1$. The threshold voltage range is thus divided into $3k+1$ bins. During reading, the bin that V_{th} falls into is determined after sensing with the reference threshold voltages, according to which the LLRs are computed. The sign of the LLR represents represents determines the value of the bit that is more likely to be, and the magnitude measures the level of confidence. We skip the detail of LLR computation here, and describe them more carefully in Section 8.

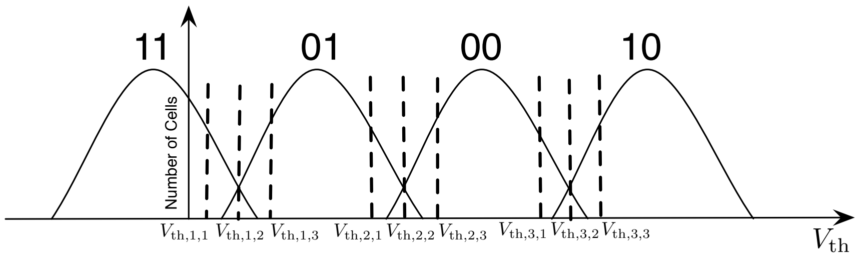


Figure 1.6: An example of soft sensing with $k = 3$.

Phase-Change Memories (PCM) has attracted significant attention in recent years [8]. PCM cells are made with chalcogenide materials. One way for configuring the cell states is through heat from electrical currents. As shown in Figure 1.7, different heating strategies result in different phases in a chalcogenide material, including the crystalline phase, the amorphous phase. Phases are distinguished by their resistance, *e.g.*, the resistance of the amorphous phase has a much higher magnitude than that of the crystalline phase. Data stored in PCM cells are represented by the corresponding phases. Compared to Flash memories, PCMs have the potential to achieve faster programming speed. This is because PCMs do not require block erasure to reset a cell state, and thus reprogramming can be made fast and cells have longer lifetime.

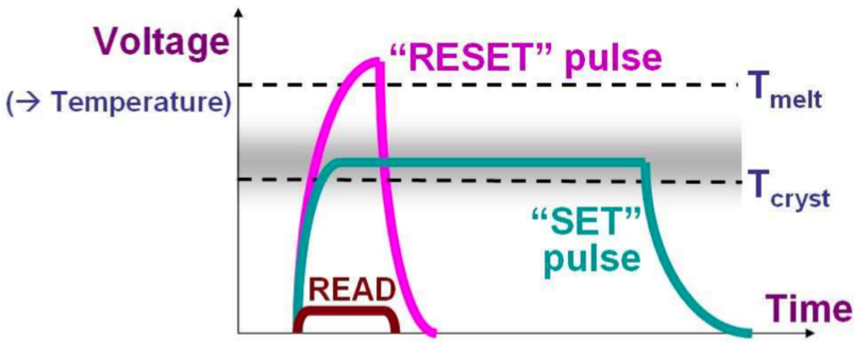


Figure 1.7: Different heating methods for operating PCM [8].

To read data stored in a PCM cell, a voltage is applied on the cell and the current flowing through the cell is measured to compute the cell resistance. Similar to flash memories, cell resistance in PCM is quantized into discrete levels, and reference threshold resistances are used for distinguishing the cell levels. The large gap between the resistance of the crystalline and the amorphous phases allows multiple intermediate phases to achieve MLCs, with more levels that provide higher storage density.

1.2 Data Reliability Challenges for Flash Memory and PCM

As the chip geometries of NAND flash and PCM continuously shrink, cells in the same block are aligned more closely, and data are more prone to errors introduced by various kinds of noise. Currently both flash memory and PCM are facing important data reliability challenges. We illustrate the major noise in flash memory and PCM in this section.

For flash memory, the errors are mainly due to *program/erase cycling (PEC)*, *overshooting* during ISPP, *cell-to-cell interference*, *charge leakage* [11], and *defective cells*. Interestingly, most of the noise introduce asymmetric errors on cell levels as we illustrate below.

- Program/erase cycling noise is caused by repetitive block erasures. When a block is erased over and over, the quality of floating gate transistors in the block degrades over time, and cells will have higher probability for not being completely erased after each block erasure, resulting some electrons stay in between the floating gate and the channel. These electrons make loose connections between the floating gate and the channel, and electrons are more easily to be injected into floating gate during programming. Therefore, program/erase cycling noise causes the means of the cell threshold voltage distributions shift to larger values. When the reference thresholds are fixed, asymmetric errors which increase cell levels will be introduced.

- Overshooting happens during ISPP. The programming device in flash is not always ideal, and too much charge may be injected into FGTs during the iterative programming. Due to the additional charge received, the cell levels after programming may become higher than the desired cell levels.

- Cell-to-cell interference happens mainly when a cell is being programmed. Due to the capacitance coupling between the adjacent cells, when a cell is being programmed, the increase of its cell threshold voltage will also raise the threshold voltages of its neighboring cells, introducing asymmetric errors by increasing the levels of its neighboring cells.

- Charge leakage occurs when data are stored in cells for a long period of time. During data retention, the electrons gradually leak away from the FGTs. Charge leakage introduces asymmetric errors which reduce the cell levels.

- Defective cells happen in imperfect manufactured cells, or cells that are worn out. Cells that are defective can not be programmed or erased. And their cell levels always stay at the same values. Defective cells cause programming errors where the actual cell level after programming may not equal to the desired level.

The noise in PCM is mainly due to the properties of phase-change materials. We describe two major noise here, namely, *resistance drift* and *cell-to-cell interference*.

- Resistance drift happens due to the structural relaxations in phase change materials [8] [30]. The resistance of a PCM cell slowly increases over time. Similar to the program/erase cycling noise in flash memories, resistance drift gradually increases the means and the variations of the cell resistance distributions. Therefore, resistance drift introduces asymmetric errors which increase the cell levels.

- Cell-to-cell interference happens during cell programming. When a cell is programmed, the heat accumulated during programming will spread to the neighboring cells. As phase-change material is very sensitive to temperature. The accumulated heat indirectly programs the neighboring cells. Such interference may change the states of those victim cells, resulting in some intermediate states depending on the current resistance of the victim cells as well as the programming operation.

In summary, the reliability of flash memories and PCM are challenged by various kinds of noise. The errors caused by each kind of noise are largely asymmetric, and may increase or decrease cell levels due to different error mechanisms. However, the errors observed by users are mixtures of different kinds of errors, which show different strength and degree of symmetry for different usages and workloads.

1.3 Overview of Research Directions

This dissertation focuses on mitigating the reliability problems of flash and PCM mentioned above. Our main tools include a set of coding techniques that are suitable for the channels and the constraints implied by flash memory and PCM such as error correcting codes, rewriting codes and codes in permutation space (which we give more background in the next chapter). We approach the problems by the following research directions.

- At bit level, we are interested in

    - Asymmetric error correcting codes for flash memory and PCM.
    - Joint coding schemes for endurance and reliability.
    - Error scrubbing methods for tracking and controlling channel quality.
    - New data representations that are inherently resisting to typical errors in flash memory and PCM.

- At higher levels, we are interested in analyzing the structures, the properties, and the meanings of the data stored in NVMs, and passing such information to the coding schemes implemented at bit level to help further improve the performance of the codes.

This dissertation is a collection of our work on mitigating the reliability issues of both flash memory and PCM. The highlights of this dissertation include the first WOM code construction which corrects a significant number of errors, a practical framework which corrects errors utilizing the redundancies in non-uniform text files, the first report of the performance of polar codes in multi-level flash chips, and the initiation on experimenting rank modulation in flash memories.

We first discuss related work in Section 2. The main contributions of this dissertation are organized as follows.

- Section 3 proposes a new coding scheme called bit-fixing codes that correct limited-magnitude errors for multi-level cell nonvolatile memories. Our codes generalize a known result and work for arbitrary error distributions.

- Section 4 designs a content-assisted file decoding framework for NVMs. Our framework uses the random access capability of NVMs and the redundancy that inherently exists in information content.

- Section 5 presents a new coding scheme that combines rewriting and error correction for the write-once memory model. Its construction is based on polar codes, and it supports any number of rewrites and corrects a substantial number of errors.

- Section 6 considers the partial information rewriting problem for flash memories. In this problem, the state of information can only be updated to a limited number of new states, and errors may occur in memory cells between two adjacent updates. We propose two coding schemes based on the models of trajectory codes. The bounds on achievable code rates are shown using polar WOM coding.

- Section 7 considers the noisy WOM model where noise is introduced during the programming. The model can be modeled with an asymmetric channel model with state information at the encoder. we propose a nesting of asymmetric channel and source polar codes which achieves the corresponding Gelfand-Pinsker bound with polynomial complexity. We also identify how the proposed technique can be used to provide a nested coding solution for other asymmetric side-information based problems.

- Section 8 reports part of the efforts towards a polar read channel for flash memories. We address several challenges raised when applying polar codes to commercial flash memories. We propose efficient schemes for shortening both non-systematic and systematic polar codes, making polar codewords be easily adapted to flash page of any length. We demonstrate that the practical performance of shortened polar codes and LDPC codes are comparable on the same data obtained by characterizing NAND MLC flash chips. We also prove the feasibility of a practical adaptive decoding framework.

- Section 9 proposes a multi-phase memory scrubbing framework for tolerating resistance drift in PCM. Our scheme refreshes a portion of the cells holding a codeword after decoding. Compared to the basic scrubbing scheme that refreshes all the cells at each scrubbing cycle, our scheme provides higher decoding frequency so that errors are detected much earlier.

- Section 10 discusses the current experimental work on emulating and characterizing rank modulation codes. Programming and reading schemes are developed to write rank modulation codewords into existing NAND flash chips under the help of a flash test board. We present the current status of the projects, preliminary experimental results, as well as discuss our future directions.

Most, if not all of the results above, have been published in our recent papers [18, 36–38, 47–49]. We discuss several future directions in Section 11.

2. BACKGROUND ON CODING FOR NONVOLATILE MEMORIES

Solutions for mitigating the reliability issues of flash memories and phase-change memories have been extensively studied in recent literature. In this section, we discuss the related work and the current progress in three directions: (1) Write-once memory (WOM) codes for rewriting data, (2) codes and schemes for correcting errors for flash memory and PCM, and (3) rank modulation codes.

2.1 WOM Codes

Coding for rewriting is an important technology for flash memories. It has the potential to substantially increase their longevity, speed and power efficiency. The most basic model for rewriting is a WOM model [61], where a set of binary cells are used to store data, and the cell levels can only increase when the data are rewritten. For flash memories, this constraint implies that the rewriting operation will delay the expensive block erasure, which leads to better preservation of cell quality and higher writing performance.

Since Rivest and Shamir's seminal work on WOM codes, the capacity region of WOM codes for noiseless channel were independently derived by Heegard [27], and by Fu and Han Vinck [23]. Towards approaching the capacity, in recent years lots of works have appeared in this area [7, 34, 35, 76, 77, 80, 81]. There have been many techniques for the design of WOM codes. They include linear code, tabular code, codes based on projective geometry, coset coding, etc. [14,54,61] Codes with substantially higher rates were discovered in recent years [76, 81]. In 2012, WOM codes that achieve capacity were discovered by Shpilka *et al.* [66,67,83] and Burshtein *et al.* [9]. The latter code used a very interesting construction based on polar coding. It should be noted that polar coding is now also used to construct rewriting codes for the rank modulation scheme [19].

Compared to the large amount of work on WOM codes, the work on WOM codes that also correct errors has been much more limited. Existing works are mainly on correcting a few errors (for example, 1, 2, or 3 errors [85, 86]). However, for rewriting to be widely used in flash memories, it is important to design WOM codes that can correct a substantial number of errors.

2.2 Error Correction Codes for Flash Memory and PCM

Multi-level cells greatly increase the storage density of nonvolatile memories. At the same time, cells are aligned more closely on the chip, and more levels need to be programmed into each cell. Such changes introduce important data reliability issues.

Error correction codes are important tools for solving the reliability issues for emerging memories such as flash memories and phase-change memories. For instance, BCH codes [50, 62] have been widely used in commodity flash memories because it has guaranteed error correction capability, and its decoder can be implemented efficiently in hardware. Low-density-parity-check (LDPC) codes [50] are also being widely studied and experimented to work for flash channels because of the great error correction performance brought by iterative decoding [60]. Recently, LDPC codes utilizing the flash channel statistics for better soft decoding is studied by Wang and Courtade [72]. Besides binary representations, error correcting codes for other data representations for flash memories are studied. For instance, Jiang and Schwartz [40] proposed error correcting codes which correct single error for the rank modulation scheme [39]. Barg and Mazumdar studied code constructions which corrects arbitrary errors for rank modulation scheme [6].

On the other hand, it was shown experimentally that errors in flash memories and phase-change memories tend to be asymmetric [82]. Such asymmetric errors may come from write disturbance, charge leakage, and resistance drift, and so on. Recently, there have been many efforts on designing new codes for correcting asymmetric errors. For instance, Ahlswede and Aydinian proposed the codes for correcting asymmetric errors [1], and optimal systematic ECCs for both asymmetric and symmetric errors were proposed by Elarief and Bose [16], codes for correcting asymmetric errors with limited magnitudes were studied by Klove and Bose [44]. Later, Cassuto and Schwartz considered asymmetric limited magnitude error corrections for flash memories to solve the overshooting problem [13]. This study was further extended by Yaakobi and Siegel to correct errors of graded magnitude, and nice results were presented [84].

Other solutions for improving the reliability of PCMs against resistance drift have been studied recently. For instance, adaptive resistance thresholds can be used to read cells more reliably [78, 88]. Designing new data representation which is robust to asymmetric errors has also been proved to be useful in practice, such as the modulation codes constructed from cell resistance ranks [39, 57]. Error logging schemes such as error correction pointers [64] are also proposed for tracking corrupted PCM cells to reduce ECCs' burden.

2.3 Rank Modulation Codes

To overcome the issues of charge leakage and overshooting problems, rank modulation codes were proposed in [39]. In this scheme, the information is carried by the relative order of cell threshold voltages rather than by their absolute cell levels. Thus, every group of cells induces a permutation, which is derived by the ranking of the level of each cell in the group. As the relative order of cell levels is used in rank modulation codes, the codes are inherently resistive to retention errors, and can be corrected without block erasure when overshooting error happens. Shortly after the work in [39], several works explored codes which correct errors in permutations specifically for the rank modulation scheme; see e.g. [6,21,41,87,89]. These works include different metrics such as Kendall's τ, Ulam, and Hamming distances. Recently, to improve the number of rewrites, the model of rank modulation was extended such that multiple cells can share the same ranking [17, 19, 42, 63]. Thus, the cells no longer determine a permutation but rather a multipermutation.

3. BIT-FIXING CODES FOR CORRECTING ASYMMETRIC ERRORS

It is important to design error-correcting codes (ECCs) that consider the many properties of MLC memories. Errors often have limited magnitudes and non-symmetric distributions, due to the memories' unique disturb and noise mechanisms. Such errors include overshooting errors, charge leakage and cell-to-cell interference. Also, current MLCs are restricted by q, the number of levels, being a power of 2. Coding schemes that can map cells to binary codes conveniently for an arbitrary number of levels are worth studying. All these will be addressed in this chapter. We provide coding schemes that adapt the general ideas of multi-level codes and multi-stage decoding [31, 71] to the channels of flash memories. The asymmetricity of flash channels allows us to design very efficient multi-stage decoding algorithms.

There are different approaches to map cell levels to binary codes when q is a power of 2, including binary representation and Gray codes. Consider n cells; and for $i = 1, \cdots, n$, let $\ell_i \in \{0, 1, \cdots, q-1\}$ be the level of the ith cell. Let $m = \log_2 q$. And let $\mathcal{B}_m(\ell_i) \triangleq (b_{i,m-1}, \cdots, b_{i,1}, b_{i,0}) \in \{0,1\}^m$ be the binary representation of ℓ_i, namely, $\ell_i = \sum_{j=0}^{m-1} b_{i,j} \cdot 2^j$. Since the m bits in a cell have different error probabilities, in a basic binary-representation approach, m ECCs of different rates are used. Specifically, for $j = 0, 1, \cdots, m-1$, we let $(b_{1,j}, \cdots, b_{n,j})$ be a separate ECC. To further reduce error probabilities, a more common approach is to represent the bits in a cell using Gray codes, and then apply m ECCs.

In this chapter, we propose an alternative coding scheme named *bit-fixing code*. Its main idea is to sequentially correct the bits in the binary representation of errors. And it can be generalized to more numeral systems. When $q = 2^m$, let $\varepsilon_i \in \{-\ell_i, \cdots, 0, \cdots, q-$

ⓒ 2012 IEEE. Reprinted, with permission, from A. Jiang, Y. Li, and J. Bruck, "Bit-fixing codes for multi-level cells," 2012 IEEE Information Theory Workshop (ITW), September, 2012.

$1 - \ell_i\}$ denote the additive error in the ith cell's level ℓ_i, and let $\mathcal{B}_m(\varepsilon_i \mod q) \triangleq (e_{i,m-1}, \cdots, e_{i,0}) \in \{0,1\}^m$ be the binary representation of $\varepsilon_i \mod q$. For $j = 0, \cdots, m-1$, let $(b_{1,j}, \cdots, b_{n,j})$ be a binary ECC $\mathcal{C}j$. The scheme has the nice property that the error bits $(e_{1,j}, \cdots, e_{n,j})$ only affect the code $\mathcal{C}_j$. (Note that this property does not hold for the binary-representation scheme introduced above.) That enables us to allocate redundancy appropriately and decode $\mathcal{C}_0$, $\cdots$, $\mathcal{C}_{m-1}$ sequentially.

The bit-fixing coding scheme can be applied to arbitrary error distributions, including both asymmetric and bidirectional errors. It can be generalized from the binary representation to many more numeral representations, including k-ary numbers (for any integer $k \geqslant 2$) and mixed-radix numeral systems such as factoradic systems. It can also be extended to an arbitrary number of cell levels, which means q can be any integer instead of a power of 2 and binary codes can still be used. The coding scheme in fact contains the ECC for asymmetric errors of limited magnitude in [13] as a special case. It is also related to the codes in [84], but is more specific in its construction and more general in various ways. It can be applied not only to storage but also to amplitude-modulation communication systems.

3.1 Bit-fixing Coding Scheme

3.1.1 Numeral Systems for Cell Levels and Errors

Consider n memory cells of q levels. For $i = 1, \cdots, n$, let $\ell_i \in \{0, 1 \cdots, q-1\}$ be the written level of the ith cell, and let $\varepsilon_i \in \{-\ell_i, \cdots, 0, \cdots, q-1-\ell_i\}$ be the additive error in the ith cell. Then the noisy cell level – the level we read – of the ith cell is $\ell'_i = \ell_i + \varepsilon_i \in \{0, 1, \cdots, q-1\}$.

Given a non-negative integer x, let $\mathcal{B}_i(x)$ denote the last i bits in the binary representation of x. That is, if $\mathcal{B}_i(x) = (y_{i-1}, \cdots, y_1, y_0) \in \{0,1\}^i$, then $(x \mod 2^i) = \sum_{j=0}^{i-1} y_j \cdot 2^j$. For example, $\mathcal{B}_3(5) = (1,0,1)$ and $\mathcal{B}_2(5) = (0,1)$. Let $m = \lceil \log_2 q \rceil$. Let

$(b_{i,m-1},\cdots,b_{i,1},b_{i,0}) \triangleq \mathcal{B}_m(\ell_i)$ be the last m bits in the binary representation of the cell level ℓ_i, and let $(e_{i,m-1},\cdots,e_{i,1},e_{i,0}) \triangleq \mathcal{B}_m(\varepsilon_i \mod 2^m)$ be the last m bits in the binary representation of $\varepsilon_i \mod 2^m$. Note that if errors are asymmetric, – say they are upward errors, namely $\forall\, i$, $\varepsilon_i \geqslant 0$, – then $(e_{i,m-1},\cdots,e_{i,1},e_{i,0}) = \mathcal{B}_m(\varepsilon_i)$.

We can extend the representation of cell levels and errors from binary representation to more general numeral systems. Let $c_1, c_2, \cdots, c_{m'}$ be positive integers such that $\prod_{i=1}^{m'} c_i \geqslant q$. Then in a mixed-radix numeral system with bases $(c_1, c_2, \cdots, c_{m'})$, every integer $x \in \{0, 1, \cdots, q-1\}$ has a unique representation $\mathcal{R}(x) = (y_{m'-1}, \cdots, y_1, y_0)$ – where for $i = 0, \cdots, m'-1$, the digit $y_i \in \{0, 1, \cdots, c_{i+1}-1\}$ – that satisfies the condition $x = y_0 + y_1 c_1 + y_2 c_1 c_2 + \cdots + y_{m'-1} c_1 c_2 \cdots c_{m'-1}$. Note that the mixed-radix numeral system contains several common numeral systems as special cases: (1) If $c_1 = c_2 = \cdots = c_{m'} = 2$, it becomes the binary representation; (2) More generally, if $c_1 = c_2 = \cdots = c_{m'} = k$ for some integer $k \geqslant 2$, it becomes the k-ary representation; (3) If $c_i = i+1$, it becomes the factorial number system. As we will see, for the bit-fixing coding scheme, given q, it is beneficial to choose the bases c_i (for $i = 1, \cdots, m'$) with two properties: First, good ECCs for alphabet size c_i can be designed; Second, the representations of errors (modulo $c_1 c_2 \cdots c_{m'}$) contain as few non-zero digits as possible.

3.1.2 Bit-fixing Coding Scheme for Binary Representation

We first present the bit-fixing coding scheme for the case where q is a power of 2 and binary representations are used. It will be extended to general cases later.

Construction 1. *Encoding of Bit-Fixing Scheme*

For $j = 0, 1, \cdots, m-1$, let $\mathcal{C}_j$ be an (n, k_j) binary ECC that can correct t_j errors. We store $k_0 + k_1 + \cdots + k_{m-1}$ information bits in n cells of $q = 2^m$ levels as follows. First, we partition the information bits into m chunks, where for $j = 0, \cdots, m-1$, the jth chunk has k_j information bits: $\mathbf{d}_j = (d_{j,1}, d_{j,2}, \cdots, d_{j,k_j})$. Next, for $j = 0, \cdots, m-1$, we use

$\mathcal{C}_j$ to encode $\mathbf{d}_j$ into a codeword $\mathbf{b}_j = (b_{1,j}, b_{2,j}, \cdots, b_{n,j})$. Then, for $i = 1, \cdots, n$, let $\ell_i = \sum_{j=0}^{m-1} b_{i,j} \cdot 2^j$, and we write the ith cell's level as ℓ_i.

After cells are written, additive errors $\varepsilon_1, \cdots, \varepsilon_n$ will appear and change cell levels to $\ell'_1 = \ell_1 + \varepsilon_1, \cdots, \ell'_n = \ell_n + \varepsilon_n$.

Construction 2. *Decoding of Bit-Fixing Scheme*

Let $\ell'_1, \cdots, \ell'_n$ be the noisy cell levels we read. As the initialization step, for $i = 1, \cdots, n$, let $\hat{\ell}_i = \ell'_i$.

For $j = 0, 1, \cdots, m-1$, carry out these three steps:

1. *For $i = 1, \cdots, n$, let $(\hat{b}_{i,m-1}, \cdots, \hat{b}_{i,1}, \hat{b}_{i,0}) = \mathcal{B}_m(\hat{\ell}_i)$ be the binary representation of the estimated cell level $\hat{\ell}_i$.*

2. *Use code $\mathcal{C}_j$ to decode the codeword $(\hat{b}_{1,j}, \cdots, \hat{b}_{n,j})$, and let $(\hat{e}_{1,j}, \cdots, \hat{e}_{n,j})$ be the discovered error vector. (That is, the recovered codeword is $(\hat{b}_{1,j} \oplus \hat{e}_{1,j}, \cdots, \hat{b}_{n,j} \oplus \hat{e}_{n,j})$, where "$\oplus$" is the exclusive-OR operation.)*

3. *For $i = 1, \cdots, n$, update the estimated cell level $\hat{\ell}_i$ as follows*

$$\hat{\ell}_i \leftarrow \left(\hat{\ell}_i - \hat{e}_{i,j} \cdot 2^j \mod q\right)$$

.

Now $\hat{\ell}_1, \cdots, \hat{\ell}_n$ are our recovered cell levels. From them, the information bits can be readily obtained.

In the above decoding algorithm, m ECCs $\mathcal{C}_0, \mathcal{C}_1, \cdots, \mathcal{C}_{m-1}$ are decoded sequentially. There is a nice mapping: The codeword of each $\mathcal{C}_i$ is $(b_{1,i}, \cdots, b_{n,i})$, which are the bits at position i in the binary representations of cell levels $(\ell_1, \cdots, \ell_n)$; and as will be shown later, the binary errors in that codeword are $(e_{1,i}, \cdots, e_{n,i})$, which are the bits at position i in the binary representations of (the modulo q of) the errors $(\varepsilon_1, \cdots, \varepsilon_n)$. However, note that the error vectors $(e_{1,i}, \cdots, e_{n,i})$ cannot be obtained directly from the binary representations of the noisy cell levels $(\ell'_1 = \ell_1 + \varepsilon_1, \cdots, \ell'_n = \ell_n + \varepsilon_n)$ (except for $i = 0$). Instead, they are obtained gradually as more and more ECCs are decoded and the estimations of cell levels are made closer and closer to their true values.

Example 1. *Consider n cells of $q = 8$ levels. Then $m = \log_2 q = 3$. Assume $\mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2$ can correct no less than 3, 1, and 2 errors, respectively. Without loss of generality (WLOG), suppose that after cells are written, errors appear in cells 1, 2 and 3, respectively. Let $\ell_1 = 3, \ell_2 = 1, \ell_3 = 2$ be their original levels, and let $\varepsilon_1 = 1, \varepsilon_2 = 5, \varepsilon_3 = -1$ be their errors. Then their noisy levels are $\ell'_1 = 4, \ell'_2 = 6, \ell'_3 = 1$, respectively. (See Figure 3.1 and the following table for an illustration.)*

	Cell 1	*Cell 2*	*Cell 3*
Original level	3: ($\underline{0}$,$\underline{1}$,$\underline{1}$)	1: ($\underline{0}$,$\underline{0}$,$\underline{1}$)	2: ($\underline{0}$,$\underline{1}$,$\underline{0}$)
Error	1: (0,0,1)	5: (1,0,1)	-1: (1,1,1)
Noisy level	4: (1,0,0)	6: (1,1,0)	1: (0,0,1)
Level after decoding $\mathcal{C}_0$	3: (0,1,$\underline{1}$)	5: (1,0,$\underline{1}$)	0: (0,0,$\underline{0}$)
Level after decoding $\mathcal{C}_1$	3: (0,$\underline{1}$,1)	5: (1,$\underline{0}$,1)	6: (1,$\underline{1}$,0)
Level after decoding $\mathcal{C}_2$	3: ($\underline{0}$,1,1)	1: ($\underline{0}$,0,1)	2: ($\underline{0}$,1,0)

In the decoding process, we first decode $\mathcal{C}_0$, where the noisy codeword is $(0, 0, 1, \cdots)$. (It is because the least-significant bits (LSB) of $(\mathcal{B}_m(\ell'_1), \mathcal{B}_m(\ell'_2), \mathcal{B}_m(\ell'_3), \cdots)$ are $(0, 0, 1, \cdots)$.) By decoding it, we find its error vector $(e_{1,0}, e_{2,0}, e_{3,0}, \cdots) = (1, 1, 1, \cdots)$. So we change the cell levels to $(4 - e_{1,0} \mod 8, 6 - e_{2,0} \mod 8, 1 - e_{3,0} \mod 8) = (3, 5, 0)$.

Next, we decode $\mathcal{C}_1$, where the noisy codeword is $(1, 0, 0, \cdots)$. (It is because the middle bits of $(\mathcal{B}_m(3), \mathcal{B}_m(5), \mathcal{B}_m(0), \cdots)$ are $(1, 0, 0, \cdots)$.) By decoding it, we find its *error vector* $(e_{1,1}, e_{2,1}, e_{3,1}, \cdots) = (0, 0, 1, \cdots)$*. So we change the cell levels to* $(3 - e_{1,1} \cdot 2 \mod 8, 5 - e_{2,1} \cdot 2 \mod 8, 0 - e_{3,1} \cdot 2 \mod 8) = (3, 5, 6)$.

We then decode $\mathcal{C}_2$*, where the noisy codeword is* $(0, 1, 1, \cdots)$*. (It is because the most-significant bits (MSB) of* $(\mathcal{B}_m(3), \mathcal{B}_m(5), \mathcal{B}_m(6), \cdots)$ *are* $(0, 1, 1, \cdots)$*.) By decoding it, we find its error vector* $(e_{1,2}, e_{2,2}, e_{3,2}, \cdots) = (0, 1, 1, \cdots)$*. So we change the cell levels to* $(3 - e_{1,2} \cdot 2^2 \mod 8, 5 - e_{2,2} \cdot 2^2 \mod 8, 6 - e_{3,2} \cdot 2^2 \mod 8) = (3, 1, 2)$*. They are the original cell levels, from which we can recover information bits.* □

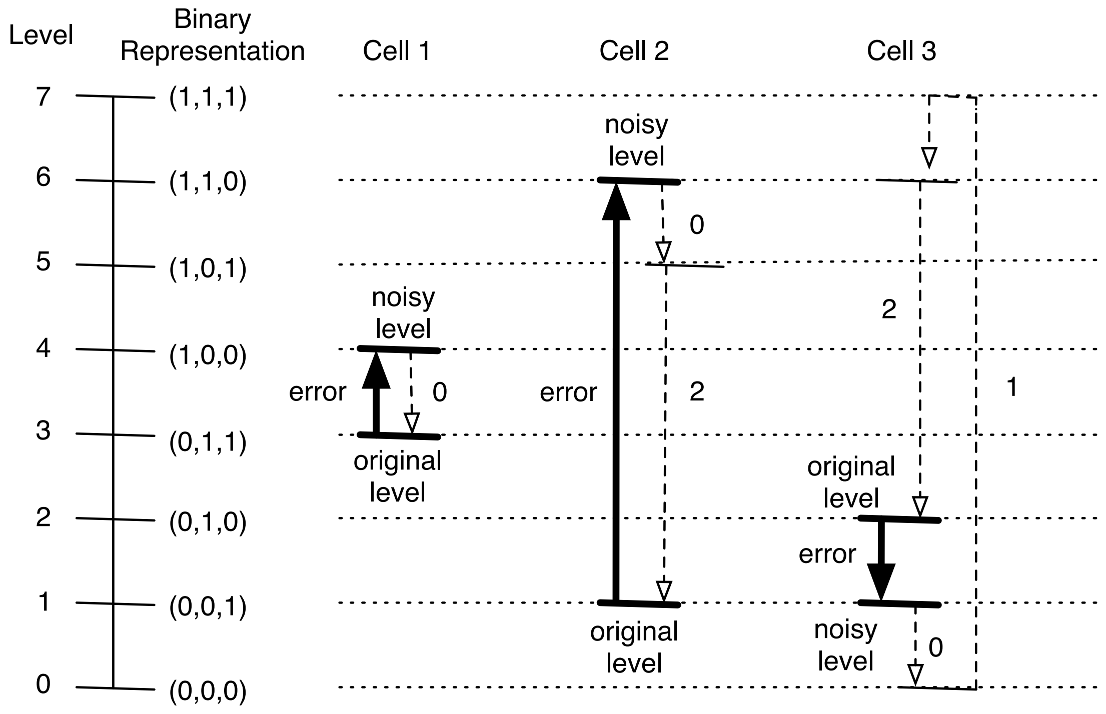


Figure 3.1: Decoding process of bit-fixing scheme. Here thick solid arrows show how errors change cell levels to noisy levels. And thin dotted arrows show how decoding changes the noisy levels back to the original levels. For $i = 0, 1, 2$, the thin dotted arrows labeled by i correspond to the decoding of $\mathcal{C}_i$.

We now prove the number of errors of variable magnitudes the bit-fixing coding scheme can correct. Given a vector $\mathbf{v} = (v_{k-1}, \cdots, v_1, v_0) \in \{0,1\}^k$, define its *support* as $support(\mathbf{v}) \triangleq \{i | i \in \{0, 1, \cdots, k-1\}, v_i = 1\}$. Given $i \in \{0, 1, \cdots, m-1\}$, we define the *cross of* i as

$$cross_m(i) \triangleq \{j | j \in \{0, 1, \cdots, 2^m - 1\}, i \in support(\mathcal{B}_m(j))\}.$$

Namely, $cross_m(i)$ is the set of integers in $\{0, 1, \cdots, 2^m - 1\}$ whose binary representations have 1 in the ith position.

Example2. *Let* $m = 3$. *Since* $\mathcal{B}_m(j) = (0,0,0)$, $(0,0,1)$, $(0,1,0)$, $(0,1,1)$, $(1,0,0)$, $(1,0,1)$, $(1,1,0)$, $(1,1,1)$ *for* $j = 0, \cdots, 7$ *respectively, we have* $cross_m(0) = \{1,3,5,7\}$, $cross_m(1) = \{2,3,6,7\}$, $cross_m(2) = \{4,5,6,7\}$. □

For $i = 0, 1, \cdots, q-1$, define $\gamma_i \triangleq |\{j | j \in \{1, \cdots, n\}, \varepsilon_j \equiv i \mod q\}|$. That is, there are γ_i cells with errors of magnitude exactly i (mod q).

Theorem3. *The bit-fixing coding scheme can recover all information bits if for* $j = 0, 1, \cdots, m-1$, *the binary error-correcting code* $\mathcal{C}_j$ *can correct* $\sum_{k \in cross_m(j)} \gamma_k$ *binary errors.*

Proof: The decoding algorithm in Construction 2 decodes $\mathcal{C}_0, \cdots, \mathcal{C}_{m-1}$ sequentially and updates the cell-level estimation $\hat{\ell}_i$ (for $i = 1, \cdots, n$) along the way. We prove this claim by induction: For $j = 0, 1, \cdots, m-1$, after $\mathcal{C}_j$ is decoded, each $\hat{\ell}_i$ is updated as $\ell_i + \sum_{k=j+1}^{m-1} e_{i,k} \cdot 2^k \mod q$.

First consider the base case $j = 0$. The noisy codeword for $\mathcal{C}_0$ is $(\ell'_1 \mod 2, \cdots, \ell'_n \mod 2)$. Since for $i = 1, \cdots, n$, $\ell'_i \equiv \ell_i + \varepsilon_i \equiv \sum_{k=0}^{m-1} b_{i,k} \cdot 2^k + \sum_{k=0}^{m-1} e_{i,k} \cdot 2^k \mod q$, the noisy codeword is $(b_{1,0} \oplus e_{1,0}, \cdots, b_{n,0} \oplus e_{n,0})$. The Hamming weight of the error vector

$(e_{1,0}, \cdots, e_{n,0})$ is $\sum_{k \in cross_m(0)} \gamma_k$, so $\mathcal{C}_0$ can correct all those errors. Then $\hat{\ell}_i$ is updated as $(\ell'_i - e_{i,0} \mod q) = (\ell_i + \sum_{k=1}^{m-1} e_{i,k} \cdot 2^k \mod q)$.

Now suppose the claim holds for $j = 0, 1, \cdots, j' < m-1$. Consider $j = j'+1$. The noisy codeword for $\mathcal{C}_j$ is $(\hat{b}_{1,j}, \cdots, \hat{b}_{n,j})$, where $\hat{b}_{i,j}$ is the $(j+1)$th LSB in the binary representation of $\hat{\ell}_i$ and equals $\left(\frac{\hat{\ell}_i - (\hat{\ell}_i \mod 2^j)}{2^j} \mod 2\right)$. By induction assumption, $\hat{\ell}_i = (\ell_i + \sum_{k=j}^{m-1} e_{i,k} \cdot 2^k \mod q) = (\sum_{k=0}^{m-1} b_{i,k} \cdot 2^k + \sum_{k=j}^{m-1} e_{i,k} \cdot 2^k \mod q)$. So $\hat{b}_{i,j} = b_{i,j} \oplus e_{i,j}$. So the noisy codeword is $(b_{1,j} \oplus e_{1,j}, \cdots, b_{n,j} \oplus e_{n,j})$, which has $\sum_{k \in cross_m(j)} \gamma_k$ errors. So $\mathcal{C}_j$ can correct all those errors. Then $\hat{\ell}_i$ is updated as $((\ell_i + \sum_{k=j}^{m-1} e_{i,k} \cdot 2^k) - e_{i,j} \cdot 2^j \mod q) = (\ell_i + \sum_{k=j+1}^{m-1} e_{i,k} \cdot 2^k \mod q)$.

So after $\mathcal{C}_0, \cdots, \mathcal{C}_{m-1}$ are all decoded, each estimated cell level $\hat{\ell}_i$ equals the original level ℓ_i, from which information bits can be recovered. $\square$

3.1.3 Bit-fixing Coding Scheme for General Numeral Systems

The above bit-fixing coding scheme can be generalized to mixed-radix numeral systems. Let m and $c_1, c_2, \cdots, c_m$ be positive integers, and let $q = c_1 c_2 \cdots c_m$. Given n cells of q levels, for $i = 1, \cdots, n$, we can represent the cell level ℓ_i as $\mathcal{R}(\ell_i) = (b_{i,m-1}, \cdots, b_{i,1}, b_{i,0}) \in \{0, \cdots, c_m - 1\} \times \cdots \times \{0, \cdots, c_2 - 1\} \times \{0, \cdots, c_1 - 1\}$ using the mixed-radix numeral system with bases $(c_1, c_2, \cdots, c_m)$. Similarly, we can represent the error ε_i as $\mathcal{R}(\varepsilon_i \mod q) = (e_{i,m-1}, \cdots, e_{i,1}, e_{i,0})$. We can then encode data in the same way as Construction 1, except that each $\mathcal{C}_j$ (for $j = 0, 1, \cdots, m-1$) is an ECC of alphabet size c_{j+1}. We can decode data in the same way as Construction 2, except that the "$\oplus$" (exclusive-OR) operation is replaced by the "mod c_{j+1}" operation and after $\mathcal{C}_j$ is decoded (for $j = 0, 1, \cdots, m-1$), the estimated cell level $\hat{\ell}_i$ is updated as $\hat{\ell}_i - \hat{e}_{i,j} \prod_{k=1}^{j} c_k \mod q$.

Given a vector $\mathbf{v} = (v_{m-1}, \cdots, v_1, v_0) \in \{0, \cdots, c_m - 1\} \times \cdots \times \{0, \cdots, c_2 - 1\} \times$

$\{0, \cdots, c_1 - 1\}$, define its *support* as

$$\mathit{support}(\mathbf{v}) \triangleq \{i | i \in \{0, 1, \cdots, m-1\}, v_i \neq 0\}.$$

Given $i \in \{0, 1, \cdots, m-1\}$, we define the *cross of* i as

$$\mathit{cross}(i) \triangleq \{j | j \in \{0, 1, \cdots, q-1\}, i \in \mathit{support}(\mathcal{R}(j))\}.$$

Theorem 4. *The bit-fixing coding scheme (for the general mixed-radix numeral system) can recover all information bits if for $j = 0, 1, \cdots, m-1$, the c_{j+1}-ary error-correcting code $\mathcal{C}_j$ can correct $\sum_{k \in cross(j)} \gamma_k$ Hamming errors.*

Note that if C_0 is an ECC and $C_1, \cdots, C_{m-1}$ contain no redundancy, the scheme here is reduced to the main code construction (Construction 1) in [**?**] for asymmetric errors of maximum magnitude $c_1 - 1$.

3.1.4 Achievable Rate of Bit-fixing Coding Scheme

We now analyze the achievable rates of the bit-fixing coding scheme. For simplicity, we present the case in which q is a power of 2 and binary representations are used. The analysis can be extended naturally to more general cases.

Consider a cell of level $\ell \in \{0, 1, \cdots, q-1\}$. Let $\ell' = \ell + \varepsilon \in \{0, 1 \cdots, q-1\}$ denote the noisy cell level, where $\varepsilon \in \{-\ell, \cdots, 0, \cdots, q-1-\ell\}$ is the error. We assume the errors in different cells are i.i.d. Due to the complex mechanisms for errors (e.g., disturbs and charge leakage, cell-level drifting and different memory manufacturing processes), it is hard to model errors with a universal model. So in this work, we consider a general model: "$\forall\ i, j \in \{0, 1, \cdots, q-1\}$, let $p_{i,j} \triangleq Pr\{\varepsilon \equiv j \mod q | \ell = i\}$ be a known distribution." Let $\mathcal{B}_m(\ell) = (b_{m-1}, \cdots, b_1, b_0)$ and $\mathcal{B}_m(\varepsilon \mod q) = (e_{m-1}, \cdots, e_1, e_0)$. Here $b_0, \cdots, b_{m-1}$ are m bits that belong to m different codes $\mathcal{C}_0, \cdots, \mathcal{C}_{m-1}$; and we let them be independent. For $i = 0, 1, \cdots, m-1$, let $\beta_i \triangleq Pr\{b_i = 0\}$; let $\alpha_{i,0} \triangleq Pr\{e_i = 1 | b_i = 0\}$ and $\alpha_{i,1} \triangleq Pr\{e_i = 1 | b_i = 1\}$ denote the cross-over probabilities in the binary channel corresponding to b_i; and define $\overline{cross}_m(i) \triangleq \{0, 1, \cdots, q-1\} - cross_m(i)$.

We can derive the cross-over probabilities:

$$Pr\{e_i = 1 | b_i = 0\} = \sum_{(d_{m-1},\cdots,d_1,d_0):\mathcal{B}_m^{-1}((d_{m-1},\cdots,d_1,d_0))\in\overline{cross}_m(i)} \sum_{k\in cross_m(i)} \left(\prod_{x\in\{0,1,\cdots,m-1\}-\{i\}} \beta_x^{1-d_x}(1-\beta_x)^{d_x} \right) \cdot p_{\mathcal{B}_m^{-1}((d_{m-1},\cdots,d_1,d_0)),k},$$

and

$$Pr\{e_i = 1 | b_i = 1\} = \sum_{(d_{m-1},\cdots,d_1,d_0):\mathcal{B}_m^{-1}((d_{m-1},\cdots,d_1,d_0))\in cross_m(i)} \sum_{k\in cross_m(i)} \left(\prod_{x\in\{0,1,\cdots,m-1\}-\{i\}} \beta_x^{1-d_x}(1-\beta_x)^{d_x} \right) \cdot p_{\mathcal{B}_m^{-1}((d_{m-1},\cdots,d_1,d_0)),k}.$$

When $n \to \infty$, for $i = 0, \cdots, m-1$, the code $\mathcal{C}_i$ can achieve rate $I(b_i; b_i \oplus e_i) = H(\beta_i(1-\alpha_{i,0}) + (1-\beta_i)\alpha_{i,1}) - \beta_i H(\alpha_{i,0}) - (1-\beta_i)H(\alpha_{i,1})$, where H is the entropy function. The bit-fixing scheme can achieve rate $\max_{\beta_0,\beta_1,\cdots,\beta_{m-1}\in[0,1]} \sum_{i=0}^{m-1} H(\beta_i(1-\alpha_{i,0}) + (1-\beta_i)\alpha_{i,1}) - \beta_i H(\alpha_{i,0}) - (1-\beta_i)H(\alpha_{i,1})$ bits per cell.

3.2 Optimal Labeling of Cell-levels

In this section, we present a new technique, *labeling of cell levels*, for better performance. So far, we have not yet differentiated the *physical state* of a cell from the *labeled level* of the cell. We have used ℓ to denote both, and the greater ℓ is, the higher the "physical state" of the cell (e.g., threshold voltage for a flash memory cell) is. However, the bit-fixing scheme presented earlier works for any labeling of cell levels. And this freedom in labeling enables the further optimization of performance. So in this section, we differentiate the physical state $s \in \{0, 1, \cdots, q-1\}$ from the labeled level $\ell \in \{0, 1, \cdots, q-1\}$ of a cell.

Let $\pi : \{0, 1, \cdots, q-1\} \rightarrow \{0, 1, \cdots, q-1\}$ be a permutation function that maps every physical state s to its corresponding level $\pi(s)$. Let $s \in \{0, 1, \cdots, q-1\}$ denote the original physical state of a cell, let $\delta \in \{-s, \cdots, 0, \cdots, q-1-s\}$ denote the physical error in it, and let $s' = s + \delta$ denote its noisy physical state. Correspondingly, let $\ell = \pi(s)$ denote its original level, let $\ell' = \pi(s')$ denote its noisy level, and let $\varepsilon = \ell' - \ell$ denote the *error in the cell level*.

The objective of a good labeling is to decrease the number of bit-errors in $\mathcal{C}_0, \cdots, \mathcal{C}_{m-1}$ caused by physical errors, and maximize the overall code rate. In the following, for simplicity, assume $q = 2^m$ and binary representations are used.

Example5. *Let* $q = 16$. *Three labelings are shown in Figure* 3.2*, which perform differently. For example, consider an error* δ *that changes the physical cell state from* $s = 5$ *to* $s' = 4$*, namely* $\delta = -1$. *In Figure* 3.2 *(a) (or (b)), with the straightforward (or Gray-code) labeling, the level is changed from* $\ell = 5$ *to* $\ell' = 4$ *(or from* $\ell = 7$ *to* $\ell' = 6$*), and the error in level is* $\varepsilon = \ell' - \ell = -1$. *Since* $\mathcal{B}_m(-1 \mod 16) = (1, 1, 1, 1)$*, 4 bit-errors are caused. In Figure* 3.2 *(c), however, the same physical error changes the level from* $\ell = 10$ *to* $\ell' = 2$*, so* $\varepsilon = \ell' - \ell = -8$. *Since* $\mathcal{B}_m(-8 \mod 16) = (1, 0, 0, 0)$*, only 1 bit-error is caused.*

Consider physical errors of magnitude one, which are very common for bidirectional errors. Since $q = 16$*, there are 30 such errors. It can be shown that* on average*, for such a physical error, the simple labeling in Figure* 3.2 *(a) introduces 2.5 bit-errors, the Gray-code labeling in Figure* 3.2 *(b) introduces 2.13 bit-errors, and the labeling in Figure* 3.2 *(c) introduces only 1.37 bit-errors.* □

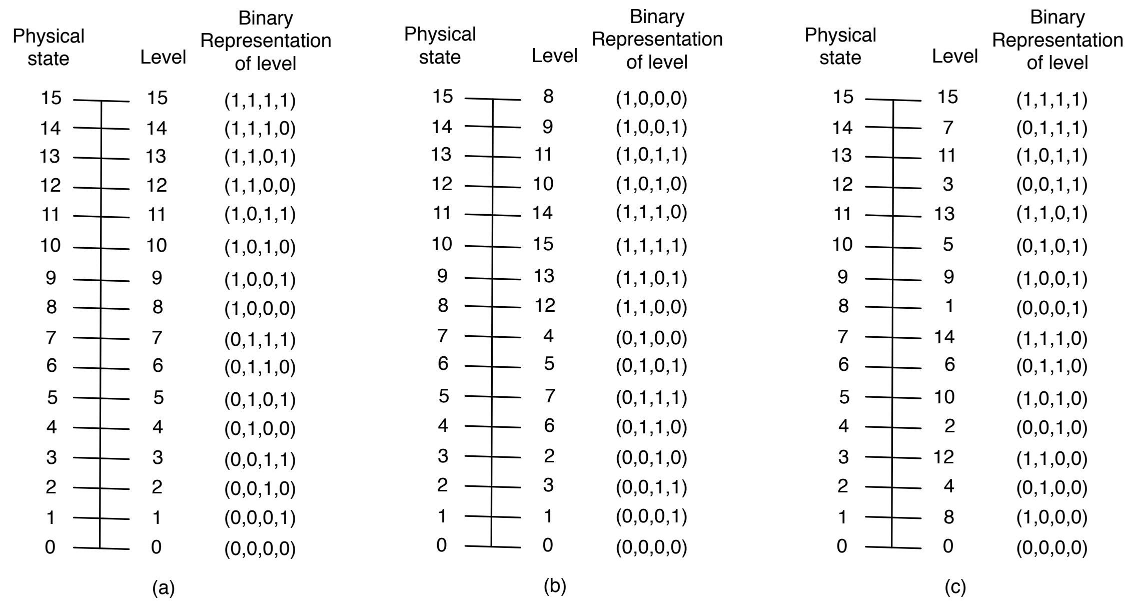


Figure 3.2: Labeling physical cell states with levels. (a) A straightforward labeling, where every physical state $s \in \{0, 1, \cdots, 15\}$ is labeled by level $\ell = s$. (b) A Gray-code labeling. (c) A new labeling.

In practice, physical errors of smaller magnitudes are usually more likely than larger ones. Let us focus on physical errors of magnitude one now, which are often the most probable errors. Given a vector $\mathbf{v} = (v_1, \cdots, v_k) \in \{0,1\}^k$, its Hamming weight is defined as $w_H(\mathbf{v}) \triangleq \{i | i \in \{1, \cdots, k\}, v_i = 1\} = |support(\mathbf{v})|$. A physical error δ changes the physical cell state from s to $s' = s + \delta$, and the number of bit-errors it introduces is $W(s,\delta) \triangleq w_H(\mathcal{B}_m(\varepsilon \mod 2^m)) = w_H(\mathcal{B}_m(\ell' - \ell \mod 2^m)) = w_H(\mathcal{B}_m(\pi(s+\delta) - \pi(s) \mod 2^m))$. Let us call a labeling π *Order-one Optimal* if it minimizes the total number of bit-errors introduced by magnitude-one (including +1 and -1) physical errors. That is, it minimizes $W_{total} \triangleq \sum_{i=0}^{q-2} W(i,1) + W(i+1,-1)$. In the following, we present such an optimal labeling.

Construction 3. *A Method for Cell-Level Labeling*

Let $\pi(0) = 0$. *For* $i = 1, 2, \cdots, m$ *and* $j = 2^{i-1}, 2^{i-1}+1, \cdots, 2^i - 1$, *let* $\pi(j) = \pi(j - 2^{i-1}) + 2^{m-i}$. □

Construction 3 generalizes Figure 3.2 (c). (Actually, it can be further generalized to the case where for $i = 0, 1, \cdots, m$ and $j = 0, 1, \cdots, 2^{m-i}-1$, $\{\pi(j \cdot 2^i + k) | k \in \{0, 1, \cdots, 2^i - 1\}\} = \{k \cdot 2^{m-i} + z \mid k \in \{0, 1, \cdots, 2^i - 1\}\}$ for some $z \in \{0, 1, \cdots, 2^{m-i} - 1\}$.) We now prove that it is order-one optimal. For any $i \in \{-2^m + 1, \cdots, -2, -1, 1, 2, \cdots, 2^m - 1\}$, define

$$\chi(i) \triangleq \max\{j | j \in \{0, 1, \cdots, m-1\}, (i \mod 2^m) \text{ is a multiple of } 2^j\}.$$

Note that in the binary representation of $(i \mod 2^m)$, – namely $\mathcal{B}_m(i \mod 2^m)$, – the $\chi(i)$ least significant bits are all 0s, and the $(\chi(i) + 1)$th least significant bit is 1. For convenience, let us define $W(2^m - 1, 1) \triangleq w_H(\mathcal{B}_m(\pi(0) - \pi(2^m - 1) \mod 2^m))$, and define $W(0, -1) \triangleq w_H(\mathcal{B}_m(\pi(2^m - 1) - \pi(0) \mod 2^m))$.

Lemma 6. *Given a labeling* $\pi : \{0, 1, \cdots, 2^m - 1\} \to \{0, 1, \cdots, 2^m - 1\}$, *for any* $i \in \{0, 1, \cdots, 2^m - 1\}$, *we have* $\chi(\pi(i+1 \mod 2^m) - \pi(i)) = \chi(\pi(i) - \pi(i+1 \mod 2^m))$ *and* $W(i, 1) + W(i+1 \mod 2^m, -1) = m - \chi(\pi(i+1 \mod 2^m) - \pi(i)) + 1$.

Corollary 7. *For* $i = 0, 1, \cdots, 2^m - 1$, *we have* $2 \leqslant W(i, 1) + W(i+1 \mod 2^m, -1) \leqslant m + 1$.

Lemma 8. *For* $j = 2, 3, \cdots, m$,

$$|\{i | 0 \leqslant i \leqslant 2^m - 1, W(i, 1) + W(i+1 \mod 2^m, -1) \leqslant j\}|$$
$$\leqslant \textstyle\sum_{k=m-1, m-2, \cdots, m-j+1} 2^k = 2^{m-j+1}(2^{j-1} - 1).$$

Theorem 9. *Construction 3 is an order-one optimal labeling.*

Sketch of proof: For $j = 2, 3, \cdots, m+1$, define

$$count(j) \triangleq |i|0 \leqslant i \leqslant 2^m - 1, W(i,1) + W(i+1 \mod 2^m, -1) = j|\,.$$

In Construction 3, $count(k) = 2^{m-k+1}$ for $k = 2, \cdots, m$, and $count(m+1) = 2$. It minimizes $\sum_{j=2}^{m+1} j \cdot count(j) - [W(2^m-1, 1) + W(0, -1))] = W_{total}$. □

3.3 Coding for Any Number of Cell Levels

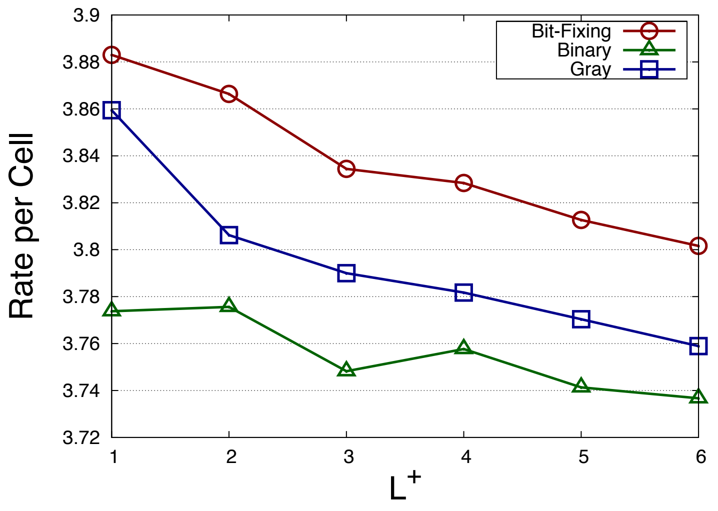
The bit-fixing scheme can be generalized to any q levels. For simplicity, we show the case where binary representations are used but q is not a power of 2. The idea is to first store data in n "virtual cells" of 2^m levels – where $m = \lceil \log_2 q \rceil$ – in the same way as before. Let $\ell^* \in \{0, 1, \cdots, 2^m - 1\}$ denote a virtual cell's level, and let $\ell \in \{0, 1, \cdots, q - 1\}$ denote the corresponding real cell's level. Then if $\ell^* \geqslant q$, we let $\ell = \ell^* - 2^{m-1}$; otherwise, we let $\ell = \ell^*$. (This *linear folding* has the property that if $\ell \neq \ell^*$, their binary representations differ only in the MSB.) It can be shown that $\mathcal{C}_0, \cdots, \mathcal{C}_{m-2}$ can be decoded the same way as before. And for $\mathcal{C}_{m-1}$, for its codeword bits, the channel model is a combination of a partial erasure channel (corresponding to the linear folding) and noise. And it can be designed and decoded accordingly.

3.4 Performance Evaluation

We have evaluated the performance of the bit-fixing scheme, and compared it to the commonly used basic binary-representation scheme and the Gray-code based scheme (which were introduced in Section I). It usually performs better than the former and is comparable to (and sometimes better than) the latter. In the following, we introduce one such comparison on achievable rates.

We consider errors of magnitude range $[-L^-, L^+]$, modeled as follows. Let there be $n \to \infty$ cells of $q = 2^m$ levels, whose errors are i.i.d. Let $p \in [0, 1]$ be a parameter, let L^+ and L^- be non-negative integers (with $L^+ + L^- > 0$), and let $\tilde{\delta} \in \{-L^-, \cdots, 0, \cdots, L^+\}$ be a random variable with this distribution: $Pr\{\tilde{\delta} = 0\} = 1 - p$; and $\forall\, i \in \{-L^-, \cdots, -1, 1, \cdots, L^+\}$, $Pr\{\tilde{\delta} = i\} = p/(L^- + L^+)$. For a cell of original physical state $s \in \{0, 1, \cdots, q-1\}$, the noise $\tilde{\delta}$ is added to it. If $\tilde{\delta} > 0$, the noisy physical level s' becomes $\min\{s + \tilde{\delta}, q-1\}$; otherwise, s' becomes $\max\{s + \tilde{\delta}, 0\}$. (It is modeled this way because a cell's state must be in $\{0, 1, \cdots, q-1\}$. And given a labeling π, the error changes the level from $\ell = \pi(s)$ to $\ell' = \pi(s')$.)

We consider the practical case where $Pr\{b_i = 0\} = 1/2$ for $i = 0, 1, \cdots, m-1$, for the reason that in most practical ECCs, codeword bits are (nearly) equally likely to be 0s and 1s. (This constraint can reduce achievable rates, however.) Some results on achievable rates are shown in Figure 9.3, with $q = 16$, $p = 0.01$, and L^+ changing from 1 to 6. Figure 9.3 (a) shows a case for asymmetric errors, where $L^- = 0$ and the bit-fixing scheme uses the simple labeling $\ell = s$. Figure 9.3 (b) shows a case for bidirectional errors, where $L^- = 3$ and the bit-fixing scheme uses the labeling in Construction 3. It can be seen that the bit-fixing coding scheme compares favorably with the basic binary-representation scheme, and is comparable to the Gray-code based scheme.



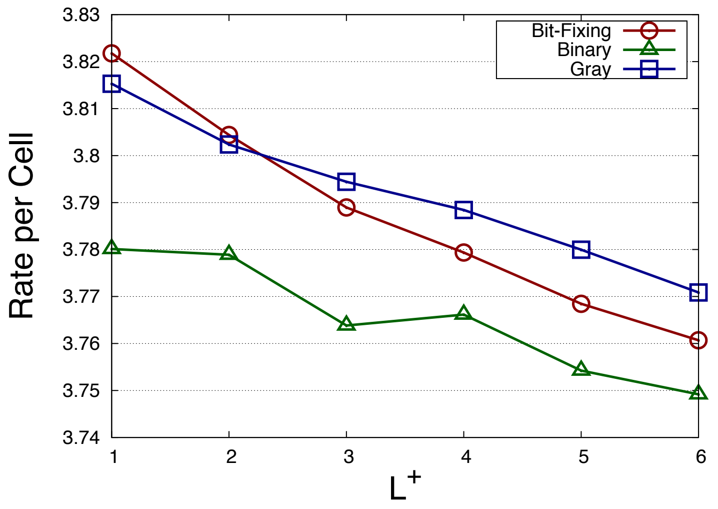


Figure 3.3: Comparison of achievable rates (number of stored bits per cell). Here $q = 16$, $p = 0.01$, and L^+ increases from 1 to 6. Above: asymmetric errors, where $L^- = 0$. Below: bidirectional errors, where $L^- = 3$.

4. CONTENT-ASSISTED ERROR CORRECTION FOR FILE RECOVERY

In this chapter, we propose a new method for error correction named *content-assisted decoding*. Our method uses the fast random access capability of NVMs and the redundancy that inherently exists in information content. Although it is theoretically possible to remove the redundancy via data compression, existing source coding algorithms do not remove all of it for efficient computation. Our method can be combined with existing storage solutions for text files. With dictionaries storing the statistical properties of words and phrases of the same language, our decoder first breaks the input noisy codeword into subcodewords, with each subcodeword corresponding to a set of possible words. The decoder then flips the bits in each noisy subcodeword to select a most likely word sequence as the correction. Consider the example in Figure 4.1. The English text *"I am"* is stored

Step	**Codeword**	**Text**
Huffman encoding	$(1,0,0,0,0,1,1,1)$	*I am*
ECC encoding	$(1,0,0,0,0,1,1,1,\mathbf{0},\mathbf{1},\mathbf{1},\mathbf{1})$	*I am*
Noise received	$(1,0,\underline{1},0,0,1,\underline{0},1,\mathbf{0},\underline{\mathbf{0}},\mathbf{1},\mathbf{1})$	*IIaa*
ECC decoding failure	$(1,0,\underline{1},0,0,1,\underline{0},1,\mathbf{0},\underline{\mathbf{0}},\mathbf{1},\mathbf{1})$	*IIaa*
Content-assisted decoding	$(1,0,0,0,0,1,1,1,\mathbf{0},\underline{\mathbf{0}},\mathbf{1},\mathbf{1})$	*I am*
ECC decoding success	$(1,0,0,0,0,1,1,1,\mathbf{0},\mathbf{1},\mathbf{1},\mathbf{1})$	*I am*

Figure 4.1: An example on correcting errors in the codeword of a text.

using the Huffman coding: $\{I \to (1,0), \sqcup \to (0,0), a \to (0,1), m \to (1,1)\}$, where $\sqcup$ denotes a space. The information bits are encoded with a $(12,8)$-shortened Hamming

© 2012 IEEE. Reprinted, with permission, from Y. Li, Y. Wang, A. Jiang, and J. Bruck, "Content-assisted file decoding for nonvolatile memories," 2012 Conference Record of the Forty Sixth Asilomar Conference on Signals, Systems and Computers (ASILOMAR), November, 2012.

code which corrects single bit errors (the bold bits denote the parity check bits). Assume that three errors (marked by the underlines) are received by the codeword. The number of errors exceeds the code's correction capability, and ECC decoding fails. Our decoder takes in the noisy codeword, and corrects the errors in the information symbols by looking up a dictionary which contains two words $\{I, am\}$. This brings the number of errors down to one. Therefore, the second trial of ECC decoding succeeds, and all the errors are corrected. Our approach is suitable for natural languages, and can potentially be extended to other types of data where the redundancy in information content is not fully removed by data compression. The scheme takes advantage of the fast random access speed provided by flash memories for fast dictionary look-up and content verification. For performance evaluation, we have tested a decoding framework that combines a soft decision decoder of low-density parity-check (LDPC) codes and our scheme with a set of text file benchmarks. Experimental results show that our decoder indeed increases the correction capability of the LDPC decoder.

The rest of the paper is organized as follows. Section 4.1 presents the preliminaries, and defines the text file decoding problem. Section 4.2 specifies the algorithms of the content-assisted file decoder. Section 4.3 discusses implementation details and experimental results.

4.1 The Models of File Decoding

We first define a few notations used throughout this chapter. Let $\mathbf{x}$ denote a binary *codeword* $(x_1, x_2, \cdots, x_n) \in \{0,1\}^n$, and we use $\mathbf{x}[i:j]$ to represent the *subcodeword* $(x_i, x_{i+1}, \cdots, x_j)$. Let the function $\mathrm{length}(\mathbf{x})$ compute the length of a codeword $\mathbf{x}$, and we use $\mathrm{d_H}(\mathbf{x}_1, \mathbf{x}_2)$ for computing the Hamming distance between two codewords of the same length. Let $\mathcal{A}$ be an alphabet set, and let $s \in \mathcal{A}$ be a symbol. We denote a *space* by $\sqcup \in \mathcal{A}$. A *word* $\mathbf{w} \triangleq (s_1, \cdots, s_n)$ of length n is a finite sequence of symbols without any space. A *phrase* $\mathbf{p} \triangleq (\mathbf{w}_1, \sqcup, \mathbf{w}_2)$ is defined as a combination of two words separated by a space. Define a text $\mathbf{t} \triangleq (\mathbf{w}_1, \sqcup, \mathbf{w}_2, \sqcup, \cdots, \sqcup, \mathbf{w}_n)$ as a sequence of words separated by $\sqcup$. A *word dictionary* $D_w \triangleq \{[\mathbf{w}_1 : p_1], [\mathbf{w}_2 : p_2], \cdots\}$ is a finite set of records where a record $[\mathbf{w} : p]$ has a key $\mathbf{w}$ and a value $p > 0$. The value p is an average probability that the word $\mathbf{w}$ occurs in a text. Similarly, a *phrase dictionary* $D_p \triangleq \{[\mathbf{p}_1 : p_1], [\mathbf{p}_2 : p_2], \cdots\}$ stores the probabilities that a set of phrases appear in any given text. The dictionary look-up operations denoted by $D_w[\mathbf{w}]$ and $D_p[\mathbf{p}]$ return the probabilities of words and phrases, respectively. We use the notation $\mathbf{w} \triangleright D_w$ (or $\mathbf{p} \triangleright D_p$) to indicate that there is a record in D_w (or D_p) with key $\mathbf{w}$ (or $\mathbf{p}$). Let π_s be a bijective mapping between a symbol and a binary codeword, and let $\mathbf{x}_s = \pi_s(\sqcup)$. In this work, the mapping π_s is used during data compression before ECC encoding, and it encodes each symbol separately. In the example show in Figure 4.1, π_s refers to the Huffman codebook. The bijective mapping between a word $\mathbf{w} = (s_1, \cdots, s_n)$ and its binary codeword is defined as $\pi_w(\mathbf{w}) \triangleq (\pi_s(s_1), \cdots, \pi_s(s_n))$, and the bijective mapping from a text to its binary representation is defined as $\pi_t(\mathbf{t}) \triangleq (\pi_w(\mathbf{w}_1), \mathbf{x}_s, \cdots, \mathbf{x}_s, \pi_w(\mathbf{w}_n))$ where $\mathbf{x}_s = \pi_s(\sqcup)$. We use π_s^{-1}, π_w^{-1} and π_t^{-1} to denote the corresponding inverse mappings.

The model of the data storage channel is shown in Figure 7.1. A text $\mathbf{t}$ is generated by

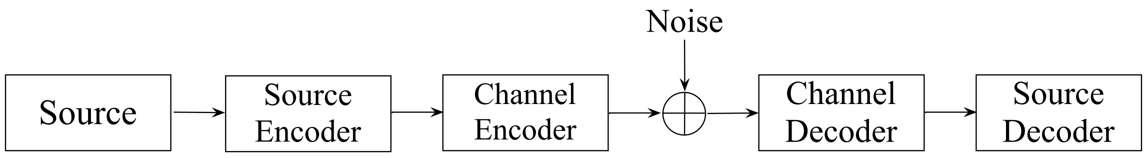


Figure 4.2: The channel model for data storage.

the source. The text is compressed by the source encoder, producing a binary codeword $\mathbf{y} = \pi_t(\mathbf{t}) \in \{0, 1\}^k$. The compressed bits are fed to a channel encoder, obtaining an ECC codeword $\mathbf{x} = \psi(\mathbf{y}) \in \{0,1\}^n$ where $n > k$. Here we assume a systematic ECC is used. The codeword is then stored by memory cells, and receives an additive error $\mathbf{e} \in \{0,1\}^n$. In this work, a binary symmetric channel (BSC) with bit-flipping rate f is assumed. When the cells are read, the channel outputs a noisy codeword $\mathbf{x}' = \mathbf{x} \oplus \mathbf{e}$ where $\oplus$ is the bit-wise exclusive-OR over codewords. The noisy codeword is first corrected by a channel decoder, producing an estimated ECC codeword $\hat{\mathbf{y}} = \psi^{-1}(\mathbf{x}')$. The source decoder decompresses the corrected codeword, and returns an estimated text $\hat{\mathbf{t}} = \pi_t^{-1}(\hat{\mathbf{y}})$ upon success.

This work focuses on designing better channel decoders ψ^{-1} for correcting bit errors in text files. We propose a new decoding framework which connects a traditional ECC decoder with a *content-assisted decoder* (CAD) as shown in Figure 4.3. A noisy codeword

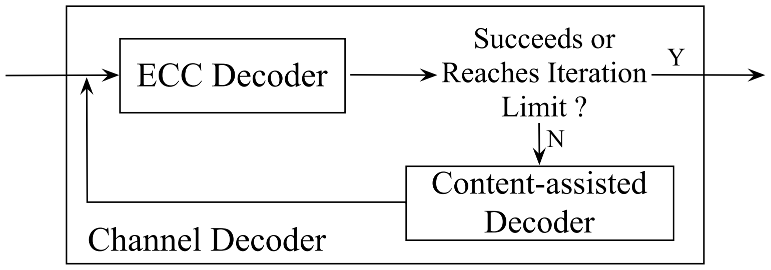


Figure 4.3: The work-flow of a channel decoder with content-assisted decoding.

is first passed into an ECC decoder. If decoding fails, the decoding output is passed to CAD. With the statistical information stored in D_w and D_p, the CAD selects a word for each subcodeword to form a likely text as the correction for the noisy codeword. The corrected text is fed back to the ECC decoder. The iteration continues until either the ECC decoder succeeds or an iteration limit is reached. The text file decoding problem for our CAD is defined as follows.

Definition 10. *Let* $\mathbf{t}$ *be some text generated from the source, and let* $\mathbf{x}' \in \{0,1\}^n$ *be a noisy channel output codeword of* $\mathbf{t}$*. Given two dictionaries* D_w *and* D_p*, the text file decoding problem for the CAD is to find an estimated text* $\hat{\mathbf{t}}$ *which is the most likely correction for* $\mathbf{x}'$, i.e.

$$\underset{\hat{\mathbf{t}}}{\operatorname{argmax}} \Pr\{\hat{\mathbf{t}} \mid \mathbf{x}', D_p, D_w\}.$$

4.2 The Content-Assisted Decoding Algorithms

The CAD approximates the solution to the problem in Definition 10 in the three steps: (1) estimate space positions in the noisy codeword to divide the codeword into subcodewords, with each subcodeword representing a set of words in D_w. (2) Resolve ambiguity by selecting a word for each subcodeword to form a most likely sequence. (3) Perform post-processing to revert the aggressive bit flips done in (1) and (2). We describe the algorithm of each step in this section.

4.2.1 Creating Dictionaries

The dictionaries D_w and D_p are used in our decoding algorithms. To create the dictionaries, we simply count the frequencies of words and phrases of two words which appear in a relatively large set of different texts in the same language as the texts generated by the source. Fast dictionary look-up is achieved by storing the dictionaries in a content-addressable way thanks to the random access in flash memories, *i.e.*, the probability in a dictionary record is addressed by the value of the corresponding word or phrase. As we show later in section 4.3, the completeness of the dictionaries effects the decoding performance.

4.2.2 Codeword Segmentation

The codeword segmentation function σ takes in a noisy codeword and a word dictionary, then flips the minimum number of bits to make the corrected codeword represent a text, *e.g.*, a sequence of valid words separated by spaces. If $\sigma(\mathbf{x}, D_w) = ((\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k), (i_1, i_2, \cdots, i_{k-1}))$, where the number of records $|D_w|$ is bounded by some constant K, and $i_j \in \mathbb{N}$ is the index of the first bit of the j-th space in $\mathbf{x}$, the subcodeword $\mathbf{x}_1 = \mathbf{x}[1 : i_1 - 1]$, $\mathbf{x}_k = \mathbf{x}[i_{k-1} + \mathrm{length}(\mathbf{x}_s) : \mathrm{length}(\mathbf{x})]$, and $\mathbf{x}_j = \mathbf{x}[i_{j-1} + \mathrm{length}(\mathbf{x}_s) : i_j - 1]$ for $j \in \{2, 3, \cdots, k-1\}$. The mapping σ is required to satisfy the following properties: (1) for each subcodeword $\mathbf{x}_j$, $\exists \mathbf{w} \triangleright D_w$ such that $\mathrm{length}(\mathbf{x}_j) = \mathrm{length}(\pi_w(\mathbf{w}))$. (2) $\mathrm{d_H}(\mathbf{x}, (\mathbf{x}_1, \mathbf{x}_s, \mathbf{x}_2, \mathbf{x}_s, \cdots, \mathbf{x}_s, \mathbf{x}_k))$ is minimized. Intuitively, as the bit-flip rate f is very small (which is common for NVM channels), the segmentation function is a maximum likelihood decoder which flips the minimum number of bits of the codeword. Let the cost function $\mathrm{c}(i, j)$ return the minimum number of flips taken to convert the subcodeword $\mathbf{x}[i : j]$ to represent a text. We have the following recurrence:

$$\mathrm{c}(i,j) \triangleq \begin{cases} \min\{\mathrm{g}(i,j), \mathrm{h}(i,j)\} & \text{if } i < j \\ \infty & \text{otherwise} \end{cases},$$

where

$$\begin{aligned} \mathrm{g}(i,j) &\triangleq \min\nolimits_{\mathbf{w} \triangleright D_w} \mathrm{d_H}(\pi_w(\mathbf{w}), \mathbf{x}[i:j]), \\ \mathrm{h}(i,j) &\triangleq \min\nolimits_{k \in [i+1, j-\mathrm{length}(\mathbf{x}_s)]} \\ &\qquad \mathrm{c}(i, k-1) + \mathrm{c}(k + \mathrm{length}(\mathbf{x}_s), j) + \\ &\qquad \mathrm{d_H}(\mathbf{x}[k : k + \mathrm{length}(\mathbf{x}_s) - 1], \mathbf{x}_s). \end{aligned}$$

The function $\mathrm{g}(i, j)$ computes the minimum number of flips taken to turn $\mathbf{x}[i : j]$ into the codeword of a word in D_w. The function $\mathrm{h}(i, j)$ computes the minimum flip cost taken to obtain a codeword representing a text with at least two words.

Example 11. *Consider the example in Figure* 4.1. *The input noisy codeword* $\mathbf{x}' =$

$(1,0,1,0,0,1,0,1)$*, and the word dictionary* $D_w = \{[I : 0.5], [am : 0.5]\}$. *We have* $\sigma(\mathbf{x}', D_w) = (((1,0),(0,1,0,1)),(3))$. *Starting from* $c(1,8)$*, we recursively compute* $\mathrm{c}(i,j)$ *for all* $i < j$. *The results are shown in Figure* 4.4b. *For instance, to compute* $\mathrm{c}(5,8)$*, we first compute* $\mathrm{g}(5,8) = 1$ *as the subcodeword can be turned to represent the word "I" with* 1 *bit-flip. We then compute* $\mathrm{h}(5,8) = \infty$. *This is because* $\mathrm{length}(\mathbf{x}_s) = 2$ *and the minimum codeword length of a word in* D_w *is* 2*, therefore it is impossible to split the subcodeword* $(0,1,0,1)$ *by a space. Finally, we have* $\mathrm{c}(5,8) = \min(1,\infty) = 1$.

Our objective is to compute $\mathrm{c}(1,n)$ given an input codeword of length n, and find out the space positions which help achieve the minimum cost. When $\mathrm{c}(i,j)$ is computed recursively starting from $\mathrm{c}(1,n)$, some entries will be recomputed unnecessarily. For instance, in example 11, the entry $\mathrm{c}(4,5)$ needs to be computed when we compute $\mathrm{c}(1,7)$ and $\mathrm{c}(2,8)$. A good way for speeding up such computation is to use dynamic programming techniques shown in Algorithm 1, which computes the final result iteratively starting from $\mathrm{c}(1,2)$, an entry computed in the previous iteration is saved for later iterations. The algorithm treats $\mathrm{c}(i,j)$ as the entries of a two dimensional table. Starting from $\mathrm{c}(1,2)$, the table the algorithm fills each entry diagonally across the table as shown in Figure 4.4a. The corresponding space locations for breaking the subcodeword $\mathbf{x}[i:j]$, or the set of words that $\mathbf{x}[i:j]$ can be flipped to represent is recorded using a two dimensional table m. In practice, as f is close to 0, the average number of errors in the codeword of a word is small. Computing the set of possible words S_w for a given noisy codeword can be accelerated by passing an additional Hamming distance limit d to reduce the search space, i.e. instead of searching the whole D_w as in $g(i,j)$, we search the set $\{\mathbf{w} \mid \mathbf{w} \triangleright D_w, \mathrm{d_H}(\pi_w(\mathbf{w}), \mathbf{x}[i:j]) < d\}$ to skip the words which are too far from the noisy codeword in terms of d and Hamming distance metric. As we are more interested in the space locations than the value of $\mathrm{c}(i,j)$, after the entries of c and m have been filled, Algorithm 2 is used to recursively trace back

Algorithm 1 CodewordSegmentation($\mathbf{x}, D_w$)

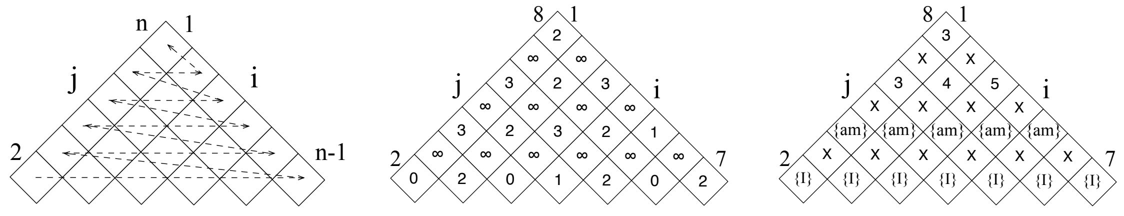
```
$n \leftarrow \mathrm{length}(\mathbf{x})$, $l \leftarrow \mathrm{length}(\mathbf{x}_s)$
Let $c$ and $m$ be two $n \times n$ tables
Let $wordSets$ and $spaces$ be two empty lists
for $t$ from $1$ to $n$ do
    for $i$ from $1$ to $n - t + 1$ do
        $j \leftarrow i + t - 1$
        $d_{\min} \leftarrow \min_{\mathbf{w} \triangleright D_w} \mathrm{d_H}(\pi_w(\mathbf{w}), \mathbf{x}[i : j])$
        $S_w \leftarrow \{\mathbf{w} \mid \mathbf{w} \triangleright D_w, \mathrm{d_H}(\pi_w(\mathbf{w}), \mathbf{x}[i : j]) = d_{\min}\}$
        $k' \leftarrow 0$
        for $k$ from $i + 1$ to $j - l$ do
            $d' \leftarrow c(i, k) + c(k + l, j) + \mathrm{d_H}(\mathbf{x}_s, \mathbf{x}[k : k + l - 1])$
            if $d' < d_{\min}$ then
                $d_{\min} \leftarrow d'$
                $k' \leftarrow k$
        if $k' = 0$ then
            $m(i, j).words \leftarrow S_w$
        else
            $m(i, j).words \leftarrow \emptyset$
            $m(i, j).space \leftarrow k'$
        $c(i, j) \leftarrow d_{\min}$
TraceBack($1, n, spaces, wordSets, m, l$)
return $wordSets$ and $spaces$
```

Algorithm 2 TraceBack($i, j, spaces, wordSets, m, l$)

```
if $m(i, j).words = \emptyset$ then
    $k \leftarrow m(i, j).space$
    TraceBack($i, k - 1, spaces, m, l$)
    $spaces$. append($k$)
    TraceBack($k + l, j, spaces, m, l$)
else
    $wordSets$. append($m(i, j).words$)
```

the solution path recorded in m. The results are the ordered space locations and the sets of words for the codewords between the spaces. Assume that K is a constant which is much smaller than N, and that the codeword of each word has limited length bounded by some constant. The time complexity of our dynamic programming algorithm is $\mathcal{O}(n)$. This is because only $\mathcal{O}(n)$ entries need to be computed and each computation takes $\mathcal{O}(1)$ time. The algorithm requires $\mathcal{O}(n^2)$ space for storing the tables c and m.

Example 12. *For the example in Figure* 4.1, *the tables c and m computed by Algorithm 1 are shown in Figure* 4.4b *and* 4.4c. *The minimum flipping cost is* $c(1,8) = 2$, *and the index of the estimated space is* $m(1,8).space = 3$. *With the estimated space, the subcodeword* $\mathbf{x}[1:2] = (1,0)$ *can be flipped to denote a word in the set* $\{I\}$, *and the subcodeword* $\mathbf{x}[5:8] = (0,1,0,1)$ *can be flipped to denote a word in the set* $\{am\}$.



(a) Iterative table filling. (b) Table c. (c) Table m.

Figure 4.4: The examples of codeword segmentation. In Figure (c): A number in the table denotes the index of the first bit of an estimated space; a set of word means the subcodeword can be flipped to any of the word in the set. The cross $\times$ means a subcodeword can neither be flipped to represent a word nor to a text with at least two words.

4.2.3 Ambiguity Resolution

Given the subcodewords $(\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k)$ between the estimated spaces, and a list of word sets $(\mathbf{W}_1, \mathbf{W}_2, \cdots, \mathbf{W}_k)$ computed from the codeword segmentation algorithm, for $i \in \{1, \cdots, k\}$ we select a word $\mathbf{w}_i$ from $\mathbf{W}_i$ to form a most probable text $\hat{\mathbf{t}} = (\mathbf{w}_1, \sqcup, \mathbf{w}_2, \sqcup, \cdots, \sqcup, \mathbf{w}_k)$. The codeword $\pi_t(\hat{\mathbf{t}})$ is a correction for the input noisy codeword. Specifically, this step is to compute

$$\begin{aligned}
&\operatorname{argmax}_{(\mathbf{w}_1,\mathbf{w}_2,\cdots,\mathbf{w}_k)\in\mathbf{W}_1\times\mathbf{W}_2\cdots\times\mathbf{W}_k} \Pr\{(\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_k) \mid (\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k)\} \\
=&\operatorname{argmax}_{(\mathbf{w}_1,\mathbf{w}_2,\cdots,\mathbf{w}_k)\in\mathbf{W}_1\times\mathbf{W}_2\cdots\times\mathbf{W}_k} \Pr\{(\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_k), (\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k)\}.
\end{aligned}$$

Let the function $\mathrm{P}(\mathbf{w}_i)$ compute the maximal joint probability when some word $\mathbf{w}_i$ is selected from $\mathbf{W}_i$ and appended to the previously selected word sequence $(\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_{i-1})$. For $i \in [2, k]$, we have

$$\begin{aligned}
\mathrm{P}(\mathbf{w}_i) \triangleq\ &\max_{(\mathbf{w}_1,\cdots,\mathbf{w}_{i-1})\in\mathbf{W}_1\times\cdots\times\mathbf{W}_{i-1}} \Pr\{(\mathbf{w}_1, \cdots, \mathbf{w}_i), (\mathbf{x}_1, \cdots, \mathbf{x}_i)\}. \\
=\ &\max_{(\mathbf{w}_1,\cdots,\mathbf{w}_{i-1})\in\mathbf{W}_1\times\cdots\times\mathbf{W}_{i-1}} \Pr\{\mathbf{w}_1\} \Pr\{\mathbf{x}_1 \mid \mathbf{w}_1\} \Pr\{\mathbf{w}_2 \mid \mathbf{w}_1\} \Pr\{\mathbf{x}_2 \mid \mathbf{w}_2\} \\
&\Pr\{\mathbf{w}_3 \mid (\mathbf{w}_1, \mathbf{w}_2)\} \Pr\{\mathbf{x}_3 \mid \mathbf{w}_3\} \cdots \Pr\{\mathbf{w}_i \mid (\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_{i-1})\} \Pr\{\mathbf{x}_i \mid \mathbf{w}_i\}
\end{aligned}$$

Assume the words in a text form a one-step Markov chain, *i.e.*, for $i \geqslant 2$, $\Pr\{\mathbf{w}_i \mid (\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_{i-1})\} = \Pr\{\mathbf{w}_i \mid \mathbf{w}_{i-1}\}$. Therefore, we rewrite the equation above as:

$$\begin{aligned}
\mathrm{P}(\mathbf{w}_i) =\ &\max_{(\mathbf{w}_1,\cdots,\mathbf{w}_{i-1})\in\mathbf{W}_1\times\cdots\times\mathbf{W}_{i-1}} \Pr\{\mathbf{w}_i \mid \mathbf{w}_{i-1}\} \Pr\{\mathbf{x}_i \mid \mathbf{w}_i\} \\
&\Pr\{\mathbf{w}_1\} \Pr\{\mathbf{w}_2 \mid \mathbf{w}_1\} \cdots \Pr\{\mathbf{w}_{i-1} \mid \mathbf{w}_{i-2}\} \prod_{k=1}^{i-1} \Pr\{\mathbf{x}_k \mid \mathbf{w}_k\} \\
=\ &\max_{(\mathbf{w}_1,\cdots,\mathbf{w}_{i-1})\in\mathbf{W}_1\times\cdots\times\mathbf{W}_{i-1}} \Pr\{\mathbf{w}_i \mid \mathbf{w}_{i-1}\} \Pr\{\mathbf{x}_i \mid \mathbf{w}_i\} \\
&\Pr\{(\mathbf{w}_1, \cdots, \mathbf{w}_{i-1}), (\mathbf{x}_1, \cdots, \mathbf{x}_{i-1})\} \qquad (4.1) \\
=\ &\max_{\mathbf{w}_{i-1}\in\mathbf{W}_{i-1}} \Pr\{\mathbf{x}_i \mid \mathbf{w}_i\} \Pr\{\mathbf{w}_i|\mathbf{w}_{i-1}\} \\
&\max_{(\mathbf{w}_1,\cdots,\mathbf{w}_{i-2})\in\mathbf{W}_1\times\cdots\times\mathbf{W}_{i-2}} \Pr\{(\mathbf{w}_1, \cdots, \mathbf{w}_{i-1}), (\mathbf{x}_1, \cdots, \mathbf{x}_{i-1})\} \\
=\ &\max_{\mathbf{w}_{i-1}\in\mathbf{W}_{i-1}} \Pr\{\mathbf{x}_i \mid \mathbf{w}_i\} \Pr\{\mathbf{w}_i|\mathbf{w}_{i-1}\} \mathrm{P}(\mathbf{w}_{i-1}).
\end{aligned}$$

and $\mathrm{P}(\mathbf{w}_1) = \Pr\{\mathbf{w}_1\} \Pr\{\mathbf{x}_1 \mid \mathbf{w}_1\}$. The conditional probability $\Pr\{\mathbf{x}_k \mid \mathbf{w}_k\}$ is computed from the channel statistics by

$$\Pr\{\mathbf{x}_k \mid \mathbf{w}_k\} = f^{\mathrm{d_H}(\pi_w(\mathbf{w}_k),\mathbf{x}_k)}(1-f)^{\mathrm{length}(\mathbf{x}_k)-\mathrm{d_H}(\pi_w(\mathbf{w}_k),\mathbf{x}_k)}.$$

The probabilities $\Pr\{\mathbf{w}_1\} = D_w[\mathbf{w}_1]$ and $\Pr\{\mathbf{w}_k \mid \mathbf{w}_{k-1}\} = D_p[(\mathbf{w}_{k-1}, \sqcup, \mathbf{w}_k)]$ are looked up from the dictionaries:

The derived recurrence suggests that the optimization problem can be mapped to the problem of trellis decoding, which is again solved by dynamic programming. The trellis for our problem has k time stages. The observed codeword at the i-th stage is $\mathbf{x}_i$ for $i \in \{1, \cdots, k\}$. There are $|\mathbf{W}_i|$ vertices at stage i with each representing an element $\mathbf{w}$ of $\mathbf{W}_i$ and being associated with the conditional probability $\Pr\{\mathbf{w} \mid \mathbf{x}_i\}$. The weight of the directed edge from a vertex at stage i with word $\mathbf{w}_x$ to a vertex of stage $i+1$ with word $\mathbf{w}_y$ is the conditional probability $\Pr\{\mathbf{w}_y \mid \mathbf{w}_x\}$. An example of the mapping is shown in Figure 4.5. Our target is to compute the sequence which achieves $\max_{\mathbf{w}_k \in \mathbf{W}_k} \mathrm{P}(\mathbf{w}_k)$, which leads to the Viterbi path in the corresponding trellis starting from a vertex in stage 1 and ending at a vertex in stage k.

The dynamic programming algorithm for solving our trellis decoding problem is specified in Algorithm 3, which is adapted from the Viterbi decoding [70]. The final solution is computed iteratively, starting from $P(\mathbf{w}_1)$ according to the recurrence. When the last iteration is finished, we trace back along the Viterbi path recorded in the table s, collecting the selected words to form an estimated text $\hat{\mathbf{t}}$. The complexity of the Viterbi decoding algorithm is $\mathcal{O}(n^2k)$ where $k = \mathcal{O}(N)$ is the length of the input codeword list, and $n = \max_{i\in[1,k]} |\mathbf{W}_i| = \mathcal{O}(K)$ is the cardinality of the biggest input word set. As K is a constant which is much smaller than N, the Viterbi decoding for our case has time complexity $O(N)$. The algorithm requires $\mathcal{O}(nk) = \mathcal{O}(N)$ space for the tables p and s.

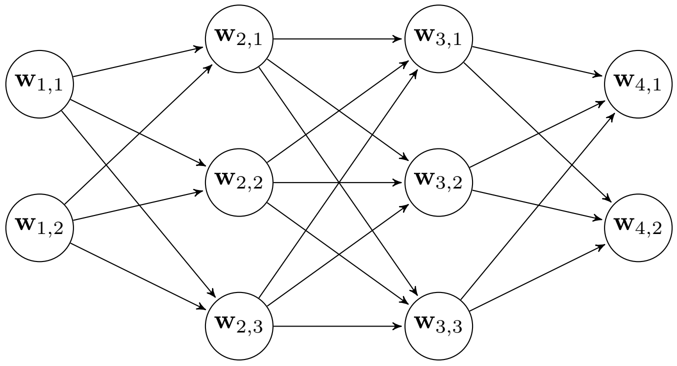


Figure 4.5: Example of the mapping to trellis decoding. The word sets $\mathbf{W}_1 = \{\mathbf{w}_{1,1}, \mathbf{w}_{1,2}\}$, $\mathbf{W}_2 = \{\mathbf{w}_{2,1}, \mathbf{w}_{2,2}, \mathbf{w}_{3,2}\}$, $\mathbf{W}_3 = \{\mathbf{w}_{3,1}, \mathbf{w}_{3,2}, \mathbf{w}_{3,3}\}$ and $\mathbf{W}_4 = \{\mathbf{w}_{4,1}, \mathbf{w}_{4,2}\}$ respectively corresponds to the subcodewords $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$ and $\mathbf{x}_4$.

Algorithm 3 Viterbi$((\mathbf{W}_1, \cdots, \mathbf{W}_k), (\mathbf{x}_1, \cdots, \mathbf{x}_k), f, D_w, D_p)$

```
n ← $\max_{l \in [1,k]} |\mathbf{W}_l|$
Let $p$ and $s$ be two $n \times k$ tables
$p_{\max} \leftarrow 0$, $index \leftarrow 0$
for $t$ from 1 to $k$ do
    for $i$ from 1 to $|\mathbf{W}_t|$ do
        $p' \leftarrow f^{\mathrm{d_H}(\pi_w(\mathbf{W}_t[i]),\mathbf{x}_t)}(1-f)^{\mathrm{length}(\mathbf{x}_t)-\mathrm{d_H}(\pi_w(\mathbf{W}_t[i]),\mathbf{x}_t)}$
        if $t = 0$ then
            $p(i,t) \leftarrow p' \cdot D_w[\mathbf{W}_t[i]]$
        else
            $p_{\max} \leftarrow 0$, $index \leftarrow 0$
            for $j$ from 1 to $|\mathbf{W}_{t-1}|$ do
                $p'' \leftarrow p' \cdot D_p[(\mathbf{W}_{t-1}[j], \sqcup, \mathbf{W}_t[i])] \cdot p[j, t-1]$
                if $p'' > p_{\max}$ then
                    $p_{\max} \leftarrow p''$
                    $index \leftarrow j$
            $p(i,t) \leftarrow p_{\max}$
            $s(i,t) \leftarrow index$
$words \leftarrow [\mathbf{W}_k[index]]$
for $t$ from $k$ to 2 do
    $i \leftarrow s(index, t)$
    $words.$ appendToFront$(\mathbf{W}_{t-1}[i])$
    $index \leftarrow i$
return $words$
```

4.2.4 Post-processing

If unknown words or phrases occur in the input codeword, additional errors will be introduced during codeword segmentation and ambiguity resolution. Unknown words (phrase) refers to new or rare words (phrases) which are not included in D_w (D_p). Upon encountering an unknown word, the codeword segmentation algorithm tends to split its codeword into subcodewords representing known words with the space symbol. Such segmentation introduces additional bit errors. We use a simple post-processing step which undoes the bit-flips issued by such aggressive segmentation. The idea is to use the phrase dictionary D_p to check whether two adjacent words returned by the Viterbi decoder is known to D_p. If so, the post-processor simply accepts the segmentation, otherwise the corresponding bits in the initial noisy codeword are used to replace the codewords for those unknown phrases. The complexity of this step is $\mathcal{O}(k) = \mathcal{O}(N)$.

4.3 Experiments

4.3.1 Implementation Detail

Our implementation supports the use of basic punctuation in the input text files, including ',', '.', '?' and '!'. This is done by adding another function in the definition of $\mathrm{c}(i, j)$ when $i < j$. The function measures the number of flips taken to turn a subcodeword to represent a word followed by a punctuation. During ambiguity resolution, overflow may occur in the multiplications of probabilities when N is large. We thus use a logarithmic version of Eq.(4.1). Using additions instead of multiplications of floating point numbers significantly delays the overflow. A smoothing technique is used for computing $\Pr\{\mathbf{w}_i \mid \mathbf{w}_{i-1}\}$. The probability $\Pr\{\mathbf{w}_i\}$ will be used if the phrase $(\mathbf{w}_{i-1}, \sqcup, \mathbf{w}_i)$ is unknown to D_p. The reason is that returning 0 for unknown phrases suddenly makes the whole joint probability be 0 in Eq.(4.1) and cancels the path.

4.3.2 Evaluation

We evaluated decoding performance of the channel decoder combining the LDPC sum-product decoder and the CAD. We compared the bit error rates (BER) of the combined channel decoder with those of the scheme using the LDPC sum-product decoding alone. The test inputs include 2 self-collected paragraphs and 8 paragraphs randomly extracted from the Canterbury Corpus, the Calgary Corpus, the Large Corpus [59], and the large text compression benchmark [53] (see Table 4.1). The dictionaries are built using the books randomly extracted from Project Gutenberg [25]. The functions π_s and π_s^{-1} are implemented with Huffman coding. A $(3584, 3141)$-random LDPC code is used as the ECC. The iteration limit of the sum-product decoder is 32. The iteration threshold for the LDPC-CAD exchange is 3. The bit-flip rate of the BSC is 0.012, which makes the sum-product decoder fail to converge with high probability. The decoding BERs for complete and

Table 4.1: The decoding BERs when the dictionaries are complete

Name	**Category**	**From**	**ECC only**	**Combined**
`email`	Email discussion	Calgary	8.6×10^{-3}	1.9×10^{-6}
`lcet`	Lecture notes	Canterbury	8.4×10^{-3}	0.0
`alice`	Novel	Canterbury	8.3×10^{-3}	2.6×10^{-6}
`confintro`	Call for paper	Self-made	8.7×10^{-3}	0.0
`bible`	The bible	Large	8.3×10^{-3}	3.2×10^{-6}
`asyoulike`	Shakespeare play	Canterbury	8.9×10^{-3}	3.8×10^{-6}
`plrabn`	Poetry	Canterbury	8.6×10^{-3}	0.0
`news`	Web news	Self-made	8.6×10^{-3}	8.4×10^{-6}
`enwiki`	Wikipedia texts	Large text	8.3×10^{-3}	0.0
`world192`	The world fact book	Large	8.3×10^{-3}	4.9×10^{-5}

incomplete dictionaries are shown in Table 4.1 and Table 4.2, respectively. The BERs for each benchmark are averaged from 1000 experiments. In Table 4.1, the combined channel decoder significantly outperforms the traditional decoder thanks to the completeness of the dictionaries. The performance for the benchmark `world192` is not as good as others. This is because `world192` has much more punctuation but fewer words than other benchmarks do, and more errors occur in the punctuations which the CAD is not good at correcting. In Table 4.2, to see the effectiveness of the post-processor, we also show the performance of the combined decoder without the post-processor. The completeness of the dictionaries determines the decoding performance. For instance, the benchmarks `world192` and `enwiki` have considerable number of words and phrases which are unknown to our dictionaries. The combined decoder without post-processing introduces additional errors by aggressively breaking the codewords of the unknown words into subcodewords separated with spaces. In such cases, the post-processor is able to recognize and revert most of the over-aggressive bit-flips. This greatly reduces the number of additional errors introduced due to the “ignorance” of the CAD. For the benchmark `confintro`, the performance of the decoder without post-processing is much better than that of the decoder using post-

Table 4.2: The decoding BERs when the dictionaries are incomplete.

Name	**ECC only**	**Combined**	**After PP**	**UW%**	**UP%**
`email`	8.6×10^{-3}	1.2×10^{-3}	6.0×10^{-4}	0	14
`lcet`	8.4×10^{-3}	9.3×10^{-4}	1.2×10^{-3}	0	24
`alice`	8.3×10^{-3}	7.6×10^{-5}	0.0	0	2
`confintro`	8.7×10^{-3}	5.1×10^{-5}	3.5×10^{-3}	0.9	41
`bible`	8.3×10^{-3}	7.5×10^{-4}	1.1×10^{-3}	0.7	29
`asyoulike`	8.9×10^{-3}	4.1×10^{-4}	9.6×10^{-4}	0.8	15
`plrabn`	8.6×10^{-3}	7.2×10^{-3}	5.0×10^{-3}	2	33
`news`	8.6×10^{-3}	1.2×10^{-3}	2.1×10^{-3}	2	29
`enwiki`	8.3×10^{-3}	1.6×10^{-2}	4.0×10^{-3}	11	34
`world192`	8.3×10^{-3}	2.6×10^{-2}	9.2×10^{-3}	25	31

processing. This is because `confintro` has only a few unknown words but many technical phrases which are unknown to D_p. The unknown phrases makes the post-processor tend to revert reasonable corrections done in the previous steps.

5. WOM CODES THAT CORRECT SYMMETRIC ERRORS

In this chapter, we present a new scheme that combines rewriting with error correction. It supports any number of rewrites and can correct a substantial number of errors. The code construction uses polar coding. Our analytical technique is based on the frozen sets corresponding to the WOM channel and the error channel, respectively, including their common degrading and common upgrading channels. We present lower bounds to the sum-rate achieved by our code. The actual sum-rates are further computed for various parameters. The analysis focuses on the binary symmetric channel (BSC). An interesting observation is that in practice, for relatively small error probabilities, the frozen set for BSC is often contained in the frozen set for the WOM channel, which enables our code to have a nested structure. The code can be further extended to multi-level cells (MLC) and more general noise models.

5.1 Basic Model

Let there be $N = 2^m$ cells that are used to store data. Every cell has two levels: 0 and 1. It can change only from level 0 to level 1, but not vice versa. That is called a WOM cell [61].

A sequence of t messages $M_1, M_2, \cdots, M_t$ will be written into the WOM cells, and when M_i is written, we do not need to remember the value of the previous messages. (Let $\mathcal{M}_j$ denote the number of bits in the message M_j, and let $M_j \in \{0,1\}^{\mathcal{M}_j}$.) For simplicity, we assume the cells are all at level 0 before the first write happens.

After cells are programmed, noise will appear in the cell levels. For now, we consider noise to be a BSC with error probability p, denoted by $BSC(p)$. These errors are hard

© 2013 IEEE. Reprinted, with permission, from A. Jiang, Y. Li, E. En Gad, M. Langberg, and J. Bruck, "Joint rewriting and error correction in write-once memories," 2013 IEEE International Symposium on Information Theory (ISIT), July, 2013.

errors, namely, they physically change the cell levels from 0 to 1 or from 1 to 0. For flash memories, such errors can be caused by read/write disturbs, interference and charge leakage, and are quite common.

5.1.1 The Model for Rewriting

A code for rewriting and error correction has t encoding functions $\mathbf{E}_1, \mathbf{E}_2, \cdots, \mathbf{E}_t$ and t decoding functions $\mathbf{D}_1, \mathbf{D}_2, \cdots, \mathbf{D}_t$. For $i = 1, 2, \cdots, N$ and $j = 1, 2, \cdots, t$, let $s_{i,j} \in \{0,1\}$ and $s'_{i,j} \in \{0,1\}$ denote the level of the i-th cell right before and after the j-th write, respectively. We require $s'_{i,j} \geqslant s_{i,j}$. Let $c_{i,j} \in \{0,1\}$ denote the level of the i-th cell at any time after the j-th write and before the $(j+1)$-th write, when reading of the message M_j can happen. The error $c_{i,j} \oplus s'_{i,j} \in \{0,1\}$ is the error in the i-th cell caused by the noise channel $BSC(p)$. (Here $\oplus$ is an XOR function.) For $j = 1, 2, \cdots, t$, the encoding function

$$\mathbf{E}_j : \{0,1\}^N \times \{0,1\}^{\mathcal{M}_j} \rightarrow \{0,1\}^N$$

changes the cell levels from $\mathbf{s}_j = (s_{1,j}, s_{2,j}, \cdots, s_{N,j})$ to $\mathbf{s}'_j = (s'_{1,j}, s'_{2,j}, \cdots, s'_{N,j})$ given the initial cell state $\mathbf{s}_j$ and the message to store M_j. (Namely, $\mathbf{E}_j(\mathbf{s}_j, M_j) = \mathbf{s}'_j$.) When the reading of M_j happens, the decoding function

$$\mathbf{D}_j : \{0,1\}^N \rightarrow \{0,1\}^{\mathcal{M}_j}$$

recovers the message M_j given the noisy cell state $\mathbf{c}_j = (c_{1,j}, c_{2,j}, \cdots, c_{N,j})$. (Namely, $\mathbf{D}_j(\mathbf{c}_j) = M_j$.)

For $j = 1, \cdots, t$, $R_j = \frac{\mathcal{M}_j}{N}$ is called the rate of the j-th write. $R_{\text{sum}} = \sum_{j=1}^{t} R_j$ is called the sum-rate of the code. When there is no noise, the maximum sum-rate of WOM code is known to be $\log_2(t+1)$; however, for noisy WOM, the maximum sum-rate is still largely unknown [27].

5.1.2 Polar Codes

We give a short introduction to polar codes due to its relevance to our code construction. A polar code is a linear block error correcting code proposed by Arıkan [2]. It is the first known code with an explicit construction that provably achieves the channel capacity of symmetric binary-input discrete memoryless channels (B-DMC). The encoder of a polar code transforms N input bits $\mathbf{u} = (u_1, u_2, \cdots, u_N)$ to N codeword bits $\mathbf{x} = (x_1, x_2, \cdots, x_N)$ through a linear transformation. (In [2], $\mathbf{x} = \mathbf{u}G_2^{\otimes m}$ where

$$G_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix},$$

and $G_2^{\otimes m}$ is the m-th Kronecker product of G_2.) The N codeword bits $(x_1, x_2, \cdots, x_N)$ are transmitted through N independent copies of a B-DMC. For decoding, N transformed binary input channels $\{W_N^{(1)}, \cdots, W_N^{(N)}\}$ can be synthesized for $u_1, u_2, \cdots, u_N$, respectively. The channels are polarized such that for large N, the fraction of indices i for which $I(W_N^{(i)})$ is nearly 1 approaches the capacity of the B-DMC [2], while the values of $I(W_N^{(i)})$ for the remaining indices i are nearly 0. The latter set of indices are called the frozen set. For error correction, the u_i's with i in the frozen set take fixed values, and the other u_i's are used as information bits. A successive cancellation (SC) decoding algorithm achieves diminishing block error probability as N increases.

Polar code can also be used for optimal lossy source coding [45], which has various applications. In particular, in [9], the idea was used to build capacity achieving WOM codes.

Our code analysis uses the concept of upgrading and degrading channels, defined based on frozen sets. As in [69], a channel $W' : X \to Z$ is called "degraded with respect to a channel $W : X \to Y$" if an equivalent channel of W' can be constructed by concatenating

W with an additional channel $Q : Y \to Z$, where the inputs of Q are linked with the outputs of W. That is,

$$W'(z|x) = \sum_{y \in Y} W(y|x)Q(z|y)$$

We denote it by $W' \preceq W$. Equivalently, the channel W is called "an upgrade with respect to W'", denoted by $W \succeq W'$.

5.2 Code Construction

In this section, we introduce our code construction that combines rewriting with error correction.

5.2.1 Basic Code Construction with a Nested Structure

5.2.1.1 Basic Concepts

First, let us consider a single rewrite step (namely, one of the t writes). Let $\mathbf{s} = (s_1, s_2, \cdots, s_N) \in \{0,1\}^N$ and $\mathbf{s}' = (s'_1, s'_2, \cdots, s'_N) \in \{0,1\}^N$ denote the cell levels right before and after this rewrite, respectively. Let $\mathbf{g} = (g_1, g_2, \cdots, g_n)$ be a pseudo-random bit sequence with i.i.d. bits that are uniformly distributed. The value of $\mathbf{g}$ is known to both the encoder and the decoder, and $\mathbf{g}$ is called a *dither*.

For $i = 1, 2, \cdots, N$, let $v_i = s_i \oplus g_i \in \{0,1\}$ and $v'_i = s'_i \oplus g_i \in \{0,1\}$ be the *value* of the i-th cell before and after the rewrite, respectively. As in [9], we build the WOM channel in Figure 5.1 for this rewrite, denoted by $WOM(\alpha, \epsilon)$. Here $\alpha \in [0,1]$ and $\epsilon \in [0, \frac{1}{2}]$ are given parameters, with $\alpha = 1 - \sum_{i=1}^{N} \frac{s_i}{N}$ representing the fraction of cells at level 0 before the rewrite, and $\epsilon = \frac{\sum_{i=1}^{N} s'_i - s_i}{N - \sum_{i=1}^{N} s_i}$ representing the fraction of cells that are changed from level 0 to level 1 by the rewrite. Let $F_{\mathrm{WOM}(\alpha,\epsilon)} \subseteq \{1, 2, \cdots, N\}$ be the frozen set of the polar code corresponding to this channel $WOM(\alpha, \epsilon)$. It is known that $\lim_{N\to\infty} \frac{|F_{\mathrm{WOM}(\alpha,\epsilon)}|}{N} = \alpha \, \mathrm{H}(\epsilon)$. [9]

For the noise channel $BSC(p)$, let $F_{\mathrm{BSC}(p)} \subseteq \{1, 2, \cdots, N\}$ be the frozen set of the

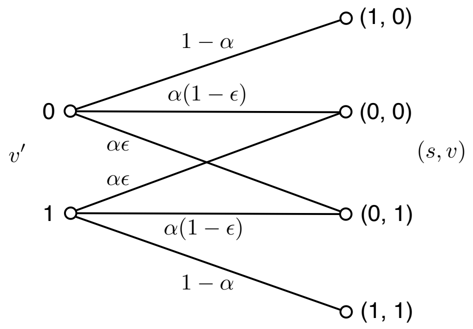


Figure 5.1: The WOM channel $WOM(\alpha, \epsilon)$.

polar code corresponding to the channel $BSC(p)$. It is known that $\lim_{N\to\infty} \frac{|F_{\text{BSC}(p)}|}{|N|} = \text{H}(p)$.

In this subsection, we assume $F_{\text{BSC}(p)} \subseteq F_{\text{WOM}(\alpha,\epsilon)}$. It is as illustrated in Figure 5.2a. In this case, the code has a nice nested structure: for any message $M \in \{0,1\}^{\mathcal{M}}$, the set of cell values $V_M \subseteq \{0,1\}^N$ that represent the message M is a linear subspace of a linear error correcting code (ECC) for the noise channel $BSC(p)$, and $\{V_M | M \in \{0,1\}^{\mathcal{M}}\}$ form a partition of the ECC. Later we will extend the code to general cases.

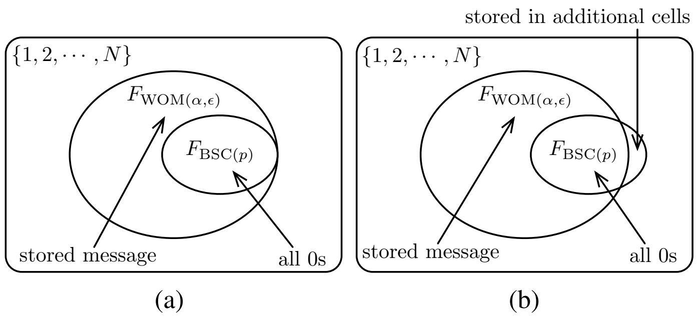


Figure 5.2: (a) Nested code for $F_{\text{BSC}(p)} \subseteq F_{\text{WOM}(\alpha,\epsilon)}$. (b) General code.

5.2.1.2 The Encoder

Let $\mathbf{E} : \{0,1\}^N \times \{0,1\}^{\mathcal{M}} \to \{0,1\}^N$ be the encoder for this rewrite. Namely, given the current cell state s and the message to write $M \in \{0,1\}^{\mathcal{M}}$, the encoder needs to find a new cell state $\mathrm{s}' = \mathbf{E}(\mathrm{s}, M)$ that represents M and is above s (that is, cell levels only increase).

The encoding process is similar to [9], but with some difference in how to assign bits to $F_{\mathrm{WOM}(\alpha,\epsilon)}$. For convenience of presentation, here we assume the polar code to be the original code designed by Arıkan [2]; however, note that it can be generalized to other polar codes as well. We present the encoding function in Algorithm 4. Here $\mathbf{y}$ and $\mathbf{u}$ are two vectors of length N; $\mathbf{u}_{F_{\mathrm{WOM}(\alpha,\epsilon)}-F_{\mathrm{BSC}(p)}} \triangleq \{u_i | i \in F_{\mathrm{WOM}(\alpha,\epsilon)} - F_{\mathrm{BSC}(p)}\}$ are all the bits u_i in the frozen set $F_{\mathrm{WOM}(\alpha,\epsilon)}$ but not $F_{\mathrm{BSC}(p)}$; $\mathbf{u}_{F_{\mathrm{BSC}(p)}} \triangleq \{u_i | i \in F_{\mathrm{BSC}(p)}\}$ are all the bits u_i in $F_{\mathrm{BSC}(p)}$; and $G_2^{\otimes m}$ is the m-th Kronecker product of G_2.

Algorithm 4 The encoding function $\mathrm{s}' = \mathbf{E}(\mathrm{s}, M)$

```
$\mathbf{y} \leftarrow ((s_1, v_1), (s_2, v_2), \cdots, (s_N, v_N))$.
Let $\mathbf{u}_{F_{\mathrm{WOM}(\alpha,\epsilon)}-F_{\mathrm{BSC}(p)}} \leftarrow M$.
Let $\mathbf{u}_{F_{\mathrm{BSC}(p)}} \leftarrow (0, 0, \cdots, 0)$.
for $i$ from $1$ to $N$ do
    if $i \notin F_{\mathrm{WOM}(\alpha,\epsilon)}$ then
        $L_N^{(i)}(\mathbf{y}, (u1, u2, \cdots, u_{i-1})) \leftarrow \frac{W_N^{(i)}(\mathbf{y},(u1,u2,\cdots,u_{i-1})|u_i=0)}{W_N^{(i)}(\mathbf{y},(u1,u2,\cdots,u_{i-1})|u_i=1)}$.
        (Comment: Here $W_N^{(i)}(\mathbf{y}, (u1, u2, \cdots, u_{i-1})|u_i = 0)$ and
        $W_N^{(i)}(\mathbf{y}, (u1, u2, \cdots, u_{i-1})|u_i = 1)$ can be computed recursively using for-
        mulae (22), (23) in [2]).
        Let $u_i \leftarrow \begin{cases} 0 & \text{with probability } \frac{L_N^{(i)}}{1+L_N^{(i)}} \\ 1 & \text{with probability } \frac{1}{1+L_N^{(i)}} \end{cases}$.
Let $\mathbf{v}' \leftarrow \mathbf{u} G_2^{\otimes m}$.
Let $\mathrm{s}' \leftarrow \mathbf{v}' \oplus \mathbf{g}$.
```

5.2.1.3 The Decoder

We now present the decoder $\mathbf{D} : \{0,1\}^N \to \{0,1\}^{\mathcal{M}}$. Let $\mathbf{c} = (c_1, c_2, \cdots, c_N) \in \{0,1\}^N$ be the noisy cell levels after the message is written. Given $\mathbf{c}$, the decoder should recover the message as $\mathbf{D}(\mathbf{c}) = M$.

Our decoder works essentially the same way as a polar error correcting code. We present it as Algorithm 5.

Algorithm 5 The decoding function $\hat{M} = \mathbf{D}(\mathbf{c})$

```
View $\mathbf{c} \oplus \mathbf{g}$ as a noisy codeword, which is the output of a binary symmetric channel $BSC(p)$. Decode $\mathbf{c} \oplus \mathbf{g}$ using the decoding algorithm of the polar error-correcting code [2], where the bits in the frozen set $F_{\mathrm{BSC}(p)}$ are set to 0s. Let $\hat{\mathbf{v}} = (\hat{v}_1, \hat{v}_2, \cdots, \hat{v}_N)$ be the recovered codeword.
Let $\hat{M} \leftarrow \left(\hat{\mathbf{v}}(G_2^{\otimes m})^{-1}\right)_{F_{\mathrm{WOM}(\alpha,\epsilon)} - F_{\mathrm{BSC}(p)}}$, which denotes the elements of the vector $\hat{\mathbf{v}}(G_2^{\otimes m})^{-1}$ whose indices are in the set $F_{\mathrm{WOM}(\alpha,\epsilon)} - F_{\mathrm{BSC}(p)}$.
```

By [2], it is easy to see that both the encoding and the decoding algorithms have time complexity $\mathcal{O}(N \log N)$.

5.2.1.4 Nested Code for t Writes

In the above, we have presented the encoder and the decoder for one rewrite. It can be naturally applied to a t-write error correcting WOM code as follows. For $j = 1, 2, \cdots, t$, for the j-th write, replace α, ϵ, $\mathbf{s}$, $\mathbf{s}'$, $\mathbf{v}$, $\mathbf{v}'$, M, $\mathcal{M}$, $\mathbf{E}$, $\mathbf{D}$, $\mathbf{c}$, $\hat{M}$, $\hat{\mathbf{v}}$ by α_{j-1}, ϵ_j, $\mathbf{s}_j$, $\mathbf{s}'_j$, $\mathbf{v}_j$, $\mathbf{v}'_j$, M_j, $\mathcal{M}_j$, $\mathbf{E}_j$, $\mathbf{D}_j$, $\mathbf{c}_j$, $\hat{M}_j$, $\hat{\mathbf{v}}_j$, respectively, and apply the above encoder and decoder.

Note that when $N \to \infty$, the values of $\alpha_1, \alpha_2, \cdots, \alpha_{t-1}$ can be computed using $\epsilon_1, \epsilon_2, \cdots, \epsilon_{t-1}$: for $BSC(p)$, $\alpha_j = \alpha_{j-1}(1-\epsilon_j)(1-p) + (1 - \alpha_{j-1}(1-\epsilon_j))p$. Optimizing the code means to choose optimal values for $\epsilon_1, \epsilon_2, \cdots, \epsilon_t$ that maximize the sum-rate.

5.2.2 Extended Code Construction

We have introduced the code for the case $F_{\text{BSC}(p)} \subseteq F_{\text{WOM}(\alpha,\epsilon)}$. Our experiments show that for relatively small p and typical values of $(\alpha_0, \epsilon_1), (\alpha_1, \epsilon_2), \cdots, (\alpha_{t-1}, \epsilon_t)$, the above condition holds. We now consider the general case where $F_{\text{BSC}(p)}$ is not necessarily a subset of $F_{\text{WOM}(\alpha,\epsilon)}$.

We first revise the encoder in Algorithm 4 as follows. After all the steps in the algorithm, we store the bits in $\mathbf{u}_{F_{\text{BSC}(p)}-F_{\text{WOM}(\alpha,\epsilon)}}$ using $N_{\text{additional},j}$ cells (for the j-th write). (It is illustrated in Figure 5.2b.) In this work, for simplicity, we assume the bits in $\mathbf{u}_{F_{\text{BSC}(p)}-F_{\text{WOM}(\alpha,\epsilon)}}$ are stored using just an error correcting code designed for the noise channel $BSC(p)$. (It will not be hard to see that we can also store it using an error-correcting WOM code, such as the one presented above, for higher rates. However, we skip the details for simplicity.) Therefore, we can have $\lim_{N\to\infty} \frac{N_{\text{additional},j}}{|F_{\text{BSC}(p)}-F_{\text{WOM}(\alpha_{j-1},\epsilon_j)}|} = \frac{1}{1-\text{H}(p)}$. And the sum-rate becomes $R_{\text{sum}} = \frac{\sum_{j=1}^{t} \mathcal{M}_j}{N+\sum_{j=1}^{t} N_{\text{additional},j}}$.

We revise the decoder in Algorithm 5. First recover the bits in $\mathbf{u}_{F_{\text{BSC}(p)}-F_{\text{WOM}(\alpha,\epsilon)}}$ using the decoding algorithm of the ECC for the $N_{\text{additional},j}$ additional cells. Then carry out all the steps in Algorithm 5, except that the bits in $F_{\text{BSC}(p)} - F_{\text{WOM}(\alpha,\epsilon)}$ are known to the decoder as the above recovered values instead of 0s.

5.3 Code Analysis for BSC

In this section, we prove the correctness of the above code construction, and analyze its performance.

5.3.1 Correctness of the Code

We first prove the correctness of our code. First, the encoder in Algorithm 4 works similarly to the WOM code encoder in [9], with an exception that the bits in $F_{\text{WOM}(\alpha,\epsilon)}$ are not all occupied by the message M; instead, the bits in its subset $F_{\text{WOM}(\alpha,\epsilon)} \cap F_{\text{BSC}(p)}$ are set to be constant values: all 0s. Therefore, it successfully rewrites data in the same way as the code in [9]. Next, the decoder in Algorithm 5 recovers the cell values from noise in the same way as the standard polar ECC. Then, the stored message M is extracted from it.

One important thing to note is that although the physical noise acts on the cell levels $\mathbf{s} = (s_1, s_2, \cdots, s_N)$, the error correcting code we use in our construction is actually for the cell values $\mathbf{v} = (v_1, v_2, \cdots, v_n) = (s_1 \oplus g_1, s_2 \oplus g_2, \cdots, s_N \oplus g_N)$. However, the pseudo-random dither $\mathbf{g}$ has independent and uniformly distributed elements; so when the noise channel for $\mathbf{s}$ is $BSC(p)$, the corresponding noise channel for $\mathbf{v}$ is also $BSC(p)$.

5.3.2 The Size of $F_{\mathrm{WOM}(\alpha,\epsilon)} \cap F_{\mathrm{BSC}(p)}$

We have seen that if $F_{\mathrm{BSC}(p)} \subseteq F_{\mathrm{WOM}(\alpha,\epsilon)}$, the code has a very interesting nested structure. In general, it is also interesting to understand how large the intersection $F_{\mathrm{WOM}(\alpha,\epsilon)} \cap F_{\mathrm{BSC}(p)}$ can be. For convenience of presentation, we consider one rewrite as in Section 5.2.1, where the parameters are α and ϵ (instead of α_{j-1}, ϵ_j).

Lemma 13. *When* $\mathrm{H}(p) \leqslant \alpha\,\mathrm{H}(\epsilon)$, $\lim_{N\to\infty} \frac{|F_{\mathrm{BSC}(p)}|}{N} \leqslant \lim_{N\to\infty} \frac{|F_{\mathrm{WOM}(\alpha,\epsilon)}|}{N}$.

Proof: $\lim_{N\to\infty} \frac{|F_{\mathrm{BSC}(p)}|}{N} = \mathrm{H}(p) \leqslant \alpha\,\mathrm{H}(\epsilon) = \lim_{N\to\infty} \frac{|F_{\mathrm{WOM}(\alpha,\epsilon)}|}{N}$. □

Lemma 14. *When* $p \leqslant \alpha\epsilon$,

$$F_{\mathrm{WOM}(\alpha,\frac{p}{\alpha})} \subseteq \left(F_{\mathrm{BSC}(p)} \cap F_{\mathrm{WOM}(\alpha,\epsilon)}\right),$$

$$\left(F_{\mathrm{WOM}(\alpha,\epsilon)} \cup F_{\mathrm{BSC}(p)}\right) \subseteq F_{\mathrm{BSC}(\alpha\epsilon)}.$$

Proof: (1) In Figure 5.3, by setting $\epsilon^* = \frac{p}{\alpha}$, we see that $BSC(p) \preceq WOM(\alpha, \frac{p}{\alpha})$. Therefore $F_{\mathrm{WOM}(\alpha,\frac{p}{\alpha})} \subseteq F_{\mathrm{BSC}(p)}$.

(2) In Figure 5.4, we can see that $WOM(\alpha,\epsilon) \preceq WOM(\alpha, \frac{p}{\alpha})$. Therefore, $F_{\mathrm{WOM}(\alpha,\frac{p}{\alpha})} \subseteq F_{\mathrm{WOM}(\alpha,\epsilon)}$.

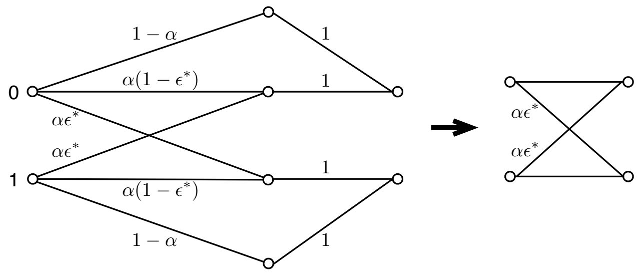


Figure 5.3: Degrading the channel $WOM(\alpha, \epsilon^*)$ to $BSC(\alpha\epsilon^*)$. The two channels on the left and on the right are equivalent.

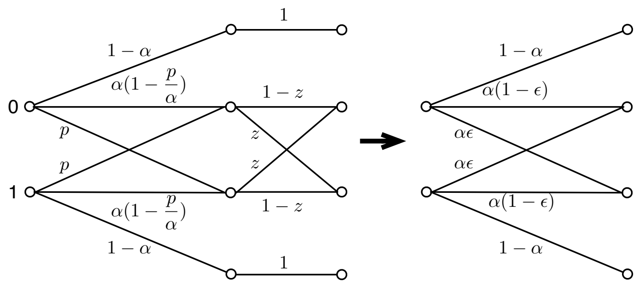


Figure 5.4: Degrading channel $WOM(\alpha, \frac{p}{\alpha})$ to $WOM(\alpha, \epsilon)$. Here $z = \frac{\alpha\epsilon - p}{\alpha - 2p}$. The two channels on the left and on the right are equivalent.

(3) In Figure 5.3, by setting $\epsilon^* = \epsilon$, we see that $BSC(\alpha\epsilon) \preceq WOM(\alpha, \epsilon)$. Therefore $F_{\mathrm{WOM}(\alpha,\epsilon)} \subseteq F_{\mathrm{BSC}(\alpha\epsilon)}$.

(4) Since $p \leqslant \alpha\epsilon$, clearly $BSC(\alpha\epsilon) \preceq BSC(p)$. Therefore $F_{\mathrm{BSC}(p)} \subseteq F_{\mathrm{BSC}(\alpha\epsilon)}$. $\square$

We illustrate the meaning of Lemma 14 in Figure 5.5.

Lemma 15. *When* $p \leqslant \alpha\epsilon$, $\lim_{N\to\infty} \frac{|F_{\mathrm{WOM}(\alpha,\epsilon)} \cap F_{\mathrm{BSC}(p)}|}{N} \geqslant \lim_{N\to\infty} \frac{|F_{\mathrm{WOM}(\alpha,\frac{p}{\alpha})}|}{N} = \alpha\,\mathrm{H}(\frac{p}{\alpha})$.

Lemma 16. *When* $p \leqslant \alpha\epsilon$, $\lim_{N\to\infty} \frac{|F_{\mathrm{WOM}(\alpha,\epsilon)} \cap F_{\mathrm{BSC}(p)}|}{N} \geqslant \lim_{N\to\infty} \frac{|F_{\mathrm{WOM}(\alpha,\epsilon)}| + |F_{\mathrm{BSC}(p)}| - |F_{\mathrm{BSC}(\alpha\epsilon)}|}{N} = \alpha\,\mathrm{H}(\epsilon) + \mathrm{H}(p) - \mathrm{H}(\alpha\epsilon)$.

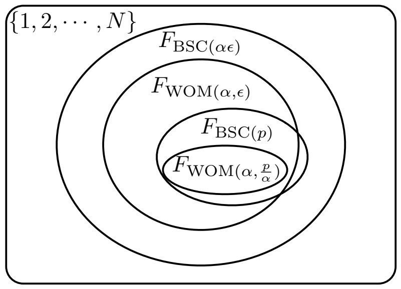

$\{1,2,\cdots,N\}$
$F_{\text{BSC}(\alpha\epsilon)}$
$F_{\text{WOM}(\alpha,\epsilon)}$
$F_{\text{BSC}(p)}$
$F_{\text{WOM}(\alpha,\frac{p}{\alpha})}$

Figure 5.5: The frozen sets for channels $BSC(p)$, $WOM(\alpha,\epsilon)$, $WOM(\alpha,\frac{p}{\alpha})$ and $BSC(\alpha\epsilon)$. Here $p \leqslant \alpha\epsilon$.

Proof: $|F_{\text{WOM}(\alpha,\epsilon)} \cap F_{\text{BSC}(p)}| = |F_{\text{WOM}(\alpha,\epsilon)}| + |F_{\text{BSC}(p)}| - |F_{\text{WOM}(\alpha,\epsilon)} \cup F_{\text{BSC}(p)}| \geqslant |F_{\text{WOM}(\alpha,\epsilon)}| + |F_{\text{BSC}(p)}| - |F_{\text{BSC}(\alpha\epsilon)}|$ (by Lemma 14). □

5.3.3 Lower Bound to Sum-rate

We now analyze the sum-rate of our general code construction as $N \to \infty$. Let $x_j \triangleq \frac{|F_{\text{WOM}(\alpha_{j-1},\epsilon_j)} \cap F_{\text{BSC}(p)}|}{|F_{\text{BSC}(p)}|} \leqslant 1$. For $j = 1, 2, \cdots, t$, the number of bits written in the j-th rewrite is

$$\begin{aligned}\mathcal{M}_j =& |F_{\text{WOM}(\alpha_{j-1},\epsilon_j)}| - |F_{\text{WOM}(\alpha_{j-1},\epsilon_j)} \cap F_{\text{BSC}(p)}| \\ =& N\alpha_{j-1}\,\mathrm{H}(\epsilon_j) - x_j|F_{\text{BSC}(p)}| \\ =& N(\alpha_{j-1}\,\mathrm{H}(\epsilon_j) - x_j\,\mathrm{H}(p))\end{aligned}$$

and the number of additional cells we use to store the bits in $F_{\text{BSC}(p)} - F_{\text{WOM}(\alpha_{j-1},\epsilon_j)}$ is

$$N_{\text{additional},j} = \frac{N\,\mathrm{H}(p)(1-x_j)}{1-\mathrm{H}(p)}$$

Therefore, the sum-rate is $R_{\text{sum}} \triangleq \frac{\sum_{j=1}^{t} \mathcal{M}_j}{N+\sum_{j=1}^{t} N_{\text{additional},j}}$

$$\begin{aligned}
=&\frac{\sum_{j=1}^{t} \alpha_{j-1} \mathrm{H}(\epsilon_j) - \mathrm{H}(p) \sum_{j=1}^{t} x_j}{1+\frac{\mathrm{H}(p)}{1-\mathrm{H}(p)} \sum_{j=1}^{t}(1-x_j)} \\
=&\frac{(1-\mathrm{H}(p)) \sum_{j=1}^{t} \alpha_{j-1} \mathrm{H}(\epsilon_j) - \mathrm{H}(p)(1-\mathrm{H}(p)) \sum_{j=1}^{t} x_j}{(1-\mathrm{H}(p)+\mathrm{H}(p)t) - \mathrm{H}(p)\sum_{j=1}^{t} x_j} \\
=&(1-\mathrm{H}(p)) \cdot \frac{\frac{1}{\mathrm{H}(p)} \sum_{j=1}^{t} \alpha_{j-1} \mathrm{H}(\epsilon_j) - \sum_{j=1}^{t} x_j}{\frac{1-\mathrm{H}(p)+\mathrm{H}(p)t}{\mathrm{H}(p)} - \sum_{j=1}^{t} x_j}.
\end{aligned}$$

Let $\gamma_j \triangleq \max\left\{ \frac{\alpha_{j-1} \mathrm{H}(\frac{p}{\alpha_{j-1}})}{\mathrm{H}(p)}, \frac{\alpha_{j-1} \mathrm{H}(\epsilon_j)+\mathrm{H}(p)-\mathrm{H}(\alpha_{j-1}\epsilon_j)}{\mathrm{H}(p)} \right\}$.

Lemma 17. *Let* $0 < p \leqslant \alpha_{j-1}\epsilon_j$. *Then* $x_j \geqslant \gamma_j$.

Proof: By Lemma 15, we have

$$\begin{aligned}
x_j =&\frac{|F_{\text{WOM}(\alpha_{j-1},\epsilon_j)} \cap F_{\text{BSC}(p)}|}{|F_{\text{BSC}(p)}|} \\
\geqslant&\frac{|F_{\text{WOM}(\alpha_{j-1},\frac{p}{\alpha_{j-1}})}|}{|F_{\text{BSC}(p)}|} = \frac{\alpha_{j-1} \mathrm{H}(\frac{p}{\alpha_{j-1}})}{\mathrm{H}(p)}.
\end{aligned}$$

By Lemma 16, we also have

$$\begin{aligned}
x_j =&\frac{|F_{\text{WOM}(\alpha_{j-1},\epsilon_j)} \cap F_{\text{BSC}(p)}|}{|F_{\text{BSC}(p)}|} \\
\geqslant&\frac{|F_{\text{WOM}(\alpha_{j-1},\epsilon_j)}| + |F_{\text{BSC}(p)}| - |F_{\text{BSC}(\alpha_{j-1}\epsilon_j)}|}{|F_{\text{BSC}(p)}|} \\
=&\frac{\alpha_{j-1} \mathrm{H}(\epsilon_j) + \mathrm{H}(p) - \mathrm{H}(\alpha_{j-1}\epsilon_j)}{\mathrm{H}(p)}.
\end{aligned}$$

□

Theorem 18 *Let* $0 < p \leqslant \alpha_{j-1}\epsilon_j$ *for* $j = 1, 2, \cdots, t$. *If* $\sum_{j=1}^{t} \alpha_{j-1} \mathrm{H}(\epsilon_j) \geqslant 1 - \mathrm{H}(p) +$

$\mathrm{H}(p)t$*, then the sum-rate* R_{sum} *is lower bounded by*

$$(1-\mathrm{H}(p))\frac{\sum_{j=1}^{t}(\alpha_{j-1}\,\mathrm{H}(\epsilon_j)-\mathrm{H}(p)\gamma_j)}{1-\mathrm{H}(p)+\mathrm{H}(p)t-\mathrm{H}(p)\sum_{j=1}^{t}\gamma_j}.$$

If $\sum_{j=1}^{t}\alpha_{j-1}\,\mathrm{H}(\epsilon_j)<1-\mathrm{H}(p)+\mathrm{H}(p)t$*, and* $\mathrm{H}(p)\leqslant\alpha_{j-1}\,\mathrm{H}(\epsilon_j)$ *for* $j=1,2,\cdots,t$*, then* R_{sum} *is lower bounded by*

$$\left(\sum_{j=1}^{t}\alpha_{j-1}\,\mathrm{H}(\epsilon_j)\right)-\mathrm{H}(p)t.$$

Proof: If $\sum_{j=1}^{t}\alpha_{j-1}\,\mathrm{H}(\epsilon_j)\geqslant 1-\mathrm{H}(p)+\mathrm{H}(p)t$, the sum-rate is minimized when x_j ($j=1,2,\cdots,t$) takes the minimum value, and we have $x_j\geqslant\gamma_j$. Otherwise, the sum-rate is minimized when x_j takes the maximum value 1. □

We show some numerical results of the lower bound to sum-rate R_{sum} in Figure 5.6, where we let $\epsilon_i=\frac{1}{2+t-i}$. The curve for $p=0$ is the optimal sum-rate for noiseless WOM code. The other four curves are the lower bounds for noisy WOM with $p=0.001$, $p=0.005$, $p=0.010$ and $p=0.016$, respectively, given by Theorem 18. Note that it is possible to further increase the lower bound values by optimizing ϵ_i. We also show in Figure 5.7 the lower bound to sum-rate when each step writes the same number of bits.

5.4 Extensions

We now consider more general noise models. For simplicity, we discuss it for an erasure channel. But it can be easily extended to other noise models. Let the noise be a BEC with erasure probability p, denoted by $BEC(p)$. After a rewrite, noise appears in some cell levels (both level 0 and level 1) and changes them to erasures. An erasure represents a noisy cell level between 0 and 1. We handle erasures this way: before a rewrite, we first increase all the erased cell levels to 1, and then perform rewriting as before.

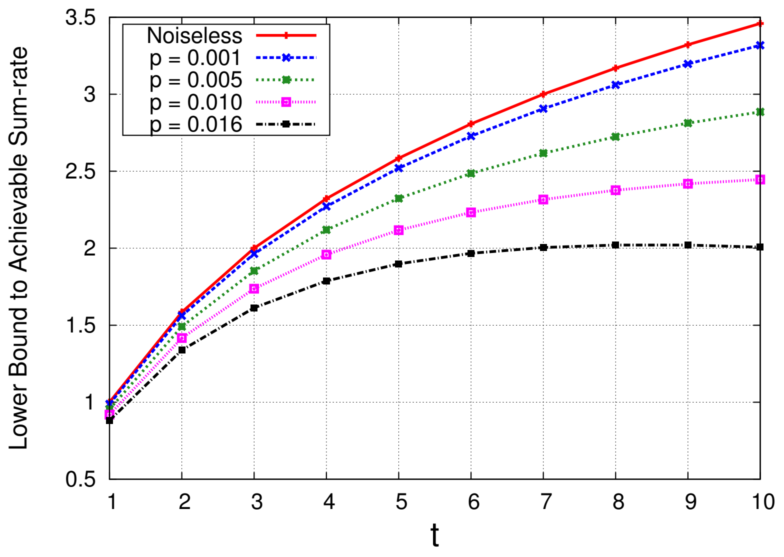


Figure 5.6: Lower bound to achievable sum-rates for different error probability p.

Note that although the noise for cell levels is $BEC(p)$, when rewriting happens, the equivalent noise channel for the cell value $\mathbf{v} = \mathbf{s} \oplus \mathbf{g}$ is a $BSC(\frac{p}{2})$, because all the erased cell levels have been pushed to level 1, and dither has a uniform distribution. Therefore, the code construction and its performance analysis can be carried out the same way as before, except that we replace p by $\frac{p}{2}$.

The code can also be extended to multi-level cells (MLC), by using q-ary polar codes. We skip the details for simplicity.

5.5 Experimental Results

In this section, we study the achievable rates of our error correcting WOM code, using polar codes of finite lengths. In the following, we assume the noise channel is $BSC(p)$, and search for good parameters $\epsilon_1, \epsilon_2, \cdots, \epsilon_t$ that achieve high sum-rate for rewriting. We also study when the code can have a nested structure, which simplifies the code construction.

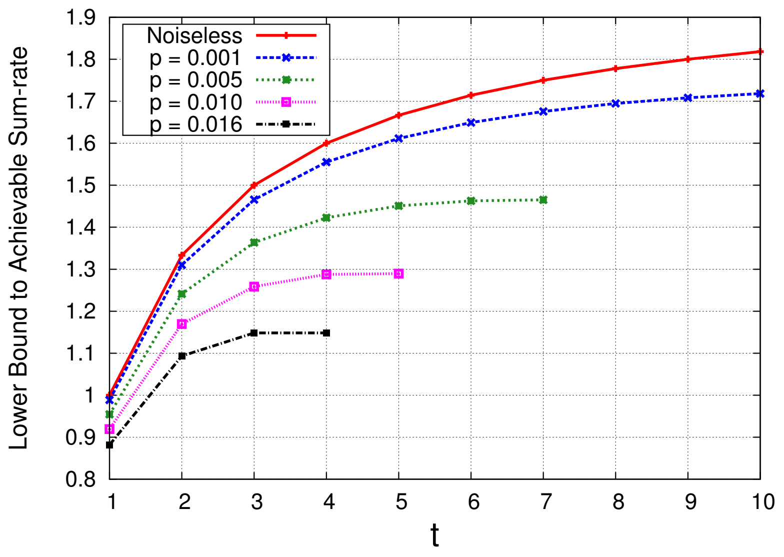


Figure 5.7: Lower bound to achievable sum-rates for different error probability p. Here each rewriting step writes the same number of bits.

5.5.1 Finding BSCs Satisfying $F_{\mathrm{BSC}(p)} \subseteq F_{\mathrm{WOM}(\alpha,\epsilon)}$

The first question we endeavor to answer is when $BSC(p)$ satisfies the condition $F_{\mathrm{BSC}(p)} \subseteq F_{\mathrm{WOM}(\alpha,\epsilon)}$, which leads to an elegant nested code structure. We search for the answer experimentally. Let $N = 8192$. Let the polar codes be constructed using the method in [69]. To obtain the frozen sets, we let $|F_{\mathrm{WOM}(\alpha,\epsilon)}| = N(\alpha\,\mathrm{H}(\epsilon) - \Delta R)$, where $\Delta R = 0.025$ is a rate loss we considered for the polar code of the WOM channel [9]; and let $F_{\mathrm{BSC}(p)}$ be chosen with the target block error rate 10^{-5}.

The results are shown in Figure 5.8. The four curves correspond to $\alpha = 0.4$, 0.6, 0.8, and 1.0, respectively. The x-axis is ϵ, and the y-axis is the maximum value of p we found that satisfies $F_{\mathrm{BSC}(p)} \subseteq F_{\mathrm{WOM}(\alpha,\epsilon)}$. Clearly, the maximum value of p increases with both α and ϵ. And it has nontrivial values (namely, it is comparable to or higher than the typical error probabilities in memories).

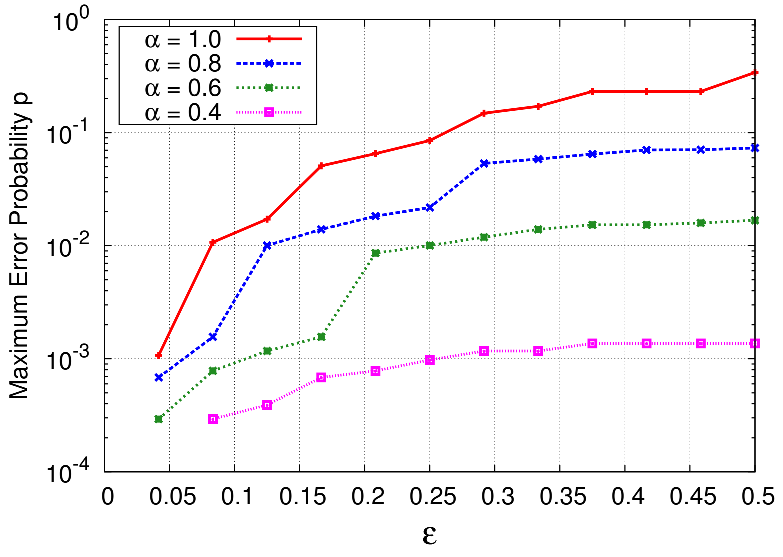


Figure 5.8: The maximum value of p found for which $F_{\mathrm{BSC}(p)} \subseteq F_{\mathrm{WOM}(\alpha,\epsilon)}$.

5.5.2 Achievable Sum-rates for Nested Code

We search for the achievable sum-rates of codes with a nested structure, namely, when the condition $F_{\mathrm{BSC}(p)} \subseteq F_{\mathrm{WOM}(\alpha_{j-1},\epsilon_j)}$ is satisfied for all $j = 1, 2, \cdots, t$. Given p, we search for $\epsilon_1, \epsilon_2, \cdots, \epsilon_t$ that maximize the sum-rate R_{sum}.

We show the results for t-write error-correcting WOM codes—for $t = 2, 3, 4, 5$—in Figure 5.9. (In the experiments, we let $N = 8192$, $\Delta R = 0.025$, and the target block error rate be 10^{-5}.) The x-axis is p, and the y-axis is the maximum sum-rate found in our algorithmic search. We see that the achievable sum-rate increases with the number of rewrites t.

5.5.3 Achievable Sum-rates for General Code

We now search for the achievable sum-rates of the general code, when $F_{\mathrm{BSC}(p)}$ is not necessarily a subset of $F_{\mathrm{WOM}(\alpha_{j-1},\epsilon_j)}$. When p is given, the general code can search a larger solution space for $\epsilon_1, \epsilon_2, \cdots, \epsilon_t$ than the nested-code case, and therefore achieve higher

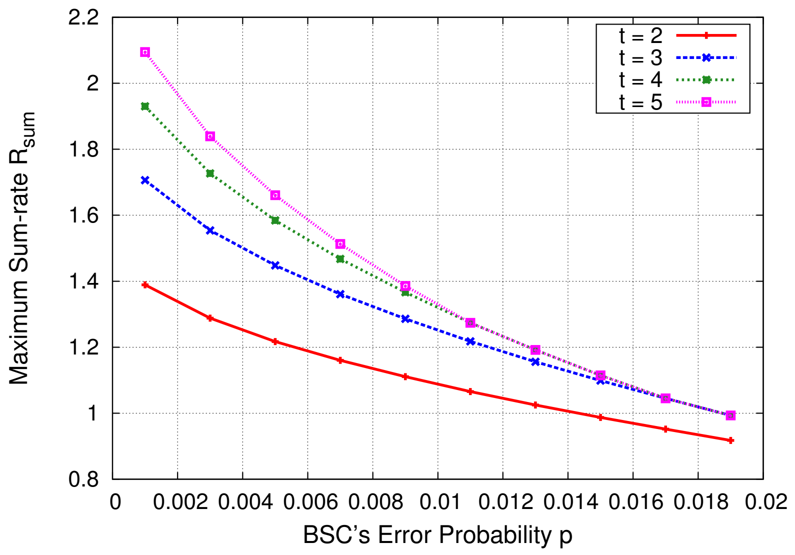


Figure 5.9: Sum-rates for different t obtained in experimental search using code length $N = 8192$, when $F_{\mathrm{BSC}(p)} \subseteq F_{\mathrm{WOM}(\alpha,\epsilon)}$.

sum-rates. However, for relatively small p (*e.g.* $p < 0.016$), the gain in rate obtained in the experiments is quite small. This means the nested code is already performing well for this parameter range. For simplicity, we skip the details.

Note that the lower bound to sum-rate R_{sum} in Figure 5.6 is actually higher than the rates we have found through experiments by now. This is because the lower bound is for $N \to \infty$, while the codes in our experiments are still short so far and consider the rate loss ΔR. Better rates can be expected as we increase the code length and further improve our search algorithm due to the results indicated by the lower bound.

6. ERROR CORRECTING TRAJECTORY CODES FOR PARTIAL INFORMATION REWRITING

In this chapter, we further generalize the results of the previous chapter to support the partial information rewriting model of trajectory codes [35], where the current information can be changed to a limited number of new states during each update.

Definition 19. *(Partial Rewriting) Let $\mathcal{G}(V, E)$ be a directed general rewriting graph that is strongly connected. Let each vertex $v \in V$ denote a message $M \in \{0,1\}^{\log_2 |V|}$ and let $\pi : \{0,1\}^{\log_2 |V|} \to V$ be a one-to-one mapping defined by $\pi(M) \triangleq v$. Let each edge $e \in E$ denote the change between the messages allowed by each update. Let D be the maximum out degree of each vertex, where $D \geqslant 1$. Partial rewriting stores a sequence of N messages $(M_0, \cdots, M_{N-1})$ such that*

(a) $\pi(M_j) \in V$ for $j \in \{0, 1, \cdots, N-1\}$.

(b) $(\pi(M_j) \to \pi(M_{j+1})) \in E$ for $j \in \{0, 1, \cdots, N-2\}$.

The model of partial rewriting can be found in many practical storage applications such as file editing, log-based file systems and file synchronization systems. In such applications, data tend to be frequently updated while each update only makes small changes on the data. Partial rewriting increases the number of block erasures in flash memories and degrades memory performance. Note that a noiseless channel is assumed in the study of trajectory code for partial rewriting [35].

The contributions of this work are general coding schemes for partial rewriting with noise, where errors from a binary symmetric channel may occur in cells between two adjacent updates. We propose two specific constructions based on trajectory codes. We show the bounds on achievable code rates based on our previous work on polar EC-WOM codes presented in Chapter 5. Our work generalize the existing rewriting codes in multiple ways.

6.1 Trajectory Codes

This section revisits the trajectory codes [35], which the codes of this chapter are mainly based on. Trajectory codes are rewriting codes that are asymptotically optimal for noiseless partial rewriting [35]. Given the rewriting graph $\mathcal{G}(V, E)$, let $L = |V|$, divide a group of n binary cells into C subgroups. For $i \in \{0, 1, \cdots, C-1\}$, let the i-th subgroup have n_i cells and be referred to as *register* r_i, namely, $n = \sum_{i=0}^{C-1} n_i$. A register stores a t-write WOM code. Let $\boldsymbol{s}_j = (s_{1,j}, s_{1,j}, \cdots, s_{n,j})$ and $\boldsymbol{s'}_j = (s'_{1,j}, s'_{1,j}, \cdots, s'_{n,j})$ be the cell states immediately before and after storing M_j. A trajectory code has Ct encoders $\mathbf{E}_0, \mathbf{E}_1, \cdots, \mathbf{E}_{Ct-1}$ and decoders $\mathbf{D}_0, \mathbf{D}_1, \cdots, \mathbf{D}_{Ct-1}$, supporting $N = Ct$ message updates. For $j \in \{0, 1, \cdots, Ct-1\}$, the encoder

$$\mathbf{E}_j : \{0,1\}^n \times \{0,1\}^{\log_2 L} \times \mathcal{G}(V, E) \to \{0,1\}^n$$

computes the new cell states from the message, the current state and $\mathcal{G}(V, E)$ (namely, $\mathbf{E}_j(\boldsymbol{s}_j, M_j, \mathcal{G}(V, E)) = \boldsymbol{s'}_j$) and the decoder

$$\mathbf{D}_j : \{0,1\}^n \times \mathcal{G}(V, E) \to \{0,1\}^{\log_2 L}$$

reads the message M_j from the current cell state at any time between the j-th and the $(j+1)$-th updates. (Namely, $\mathbf{D}_j(\boldsymbol{s'}_j, \mathcal{G}(V, E)) = M_j$.)

The Ct updates are performed using a differential scheme: the first message M_0 is stored in r_0. To write message M_1, we compute the label $\Delta_1 \in \{0,1\}^{\log_2 D}$ of the edge $\pi(M_0) \to \pi(M_1)$ in $\mathcal{G}(V, E)$, and store in r_1. (Instead of labeling edges globally, each outgoing edge of a vertex is given a local label that costs $\log_2 D$ bits, where D is the *maximum out degree*.) The next $C-2$ updates can be written in the same way. After r_{C-1} is used, an update cycle is completed, and the register r_0 will be rewritten with the new $\log L$-bit message for the next update. The iteration continues until the last update is finished. The construction implies the constraint that for all j, and for all i such that the i-th cell belongs to $r_{j \bmod C}$, we have $s'_{i,j} \geqslant s_{i,j}$. The code rate of trajectory codes is

$$R \triangleq \frac{Ct \log_2 L}{n} \text{ bits/cell.} \tag{6.1}$$

6.2 Error Correcting Trajectory Codes

We study the coding problem for joint partial rewriting and error correction, where the partial rewriting model is extended by allowing cell states to be changed by noise between two adjacent updates. In flash memories, the noise is from various sources such as interference and charge leakage [11].

6.2.1 Error Model and WOM Parameters

Before we present the code construction, we first introduce the related model and parameters. Let the noise channel for the errors received by a register between two adjacent updates (e.g. the time period after storing M_3 and before storing M_4) be a binary symmetric channel $\mathrm{BSC}(p)$ with $p \in (0, \frac{1}{2})$. We assume that errors start occurring in a register after the register is written for the first time. The assumption is motivated by practical flash memories, where the major errors for rewriting is introduced by cell-to-cell interference that happens mainly when cells are being programmed [11]. Following the model of trajectory codes, the noise channel that a register goes through at the time immediately before its next WOM rewrite is $\mathrm{BSC}(p^{*C})$, where p^{*C} is the overall error probability of C cascaded $\mathrm{BSC}(p)$ computed using $p^{*i} \triangleq \frac{1-(1-2p)^i}{2}$.

In WOM, it is common to use some parameters to control the amount of information that is written in each write. For $j \in [t]$, let the parameter $\alpha_{i,j-1}$ be the fraction of cells that are at state 0 immediately before the j-th write of the register r_i's WOM code. We have $\alpha_{i,0} = 1$. Let the parameter $\epsilon_{i,j} \leqslant \frac{1}{2}$ be the fraction of cells at state 0 that will be raised to 1 by the j-th rewrite. We have $\epsilon_{i,t} = \frac{1}{2}$. The parameters of the WOM codes used in our setting of partial rewriting also depends on the error probability p. When $n_i \to \infty$, the values of $\alpha_{i,1}$, $\alpha_{i,2}$, $\cdots$, $\alpha_{i,t-1}$ are computed by $\alpha_{i,j} = [\alpha_{i,j-1}(1 - \epsilon_{i,j})] * p^{*C}$, where $a * b \triangleq a(1-b) + (1-a)b$, and the parameters $\epsilon_{i,1}, \epsilon_{i,2}, \cdots, \epsilon_{i,t}$ are specified by users.

6.2.2 Code Construction

Our first construction is a natural extension of trajectory codes, where each register independently corrects the errors in it. The recovered messages are used by the next update. We formally present the construction by defining the encoder and the decoder in the following.

Construction 4. *For $i \in \{0, 1, \cdots, C-1\}$, let register r_i use a t-write EC-WOM code, correcting the errors from $\mathrm{BSC}(p^{*(C-i)})$. Let $l = j \bmod C$, and let $\boldsymbol{s}_j$ be the states of the n cells right before the j-th update, and let $\boldsymbol{s'}_j$ denote the cell states at any time between the j-th and the $(j+1)$-th update. (Therefore, $\boldsymbol{s}_j$ is the value of $\boldsymbol{s'}_{j-1}$ at a particular moment.) For $j = 0, 1, \cdots, Ct-1$, we have*

Encoder $\mathbf{E}_j(\boldsymbol{s}_j, M_j, \mathcal{G}(V,E)) = \boldsymbol{s'}_j$

If $l = 0$, rewrite r_0 with M_j. Otherwise, do;

(1) Recover message $\hat{M}_{j-1} = \mathbf{D}_{j-1}(\boldsymbol{s}_j, \mathcal{G}(V,E))$.

(2) Compute the label Δ_j s.t. $(\pi(\hat{M}_{j-1}) \xrightarrow{\Delta_j} \pi(M_j)) \in E$. (Here π is specified in Definition 1.)

(3) Store the label Δ_j in register r_l using rewriting (i.e. using the EC-WOM code for r_l).

Decoder $\mathrm{D}_j(\boldsymbol{s'}_j, \mathcal{G}(V,E)) = \hat{M}_j$

(1) Decode r_0 and obtain the estimated message $\hat{M}_{j-l}$. Let $\hat{v}_{j-l} = \pi(\hat{M}_{j-l})$.

(2) For k from 1 to l, decode r_k and obtain the estimated edge label $\hat{\Delta}_{j-l+k}$. (Note that in the rewriting graph $\mathcal{G}(V,E)$, the edge from message $M_{j-l+k-1}$ to message $M_{j-l+k-1}$ has the label Δ_{j-l+k}).

(3) Compute $\hat{M}_j$. Start from the vertex $\hat{v}_{j-l}$, traverse $\mathcal{G}(V,E)$ along the path marked by $\hat{\Delta}_{j-l+1}, \hat{\Delta}_{j-l+2}, \cdots, \hat{\Delta}_j$, which leads to $\hat{v}_j$. Output $\hat{M}_j = \pi^{-1}(\hat{v}_j)$.

Example 20. *We now show a simple example for $n = 6$ cells. (In practice, the code usually has thousands of cells.) Let the cells be divided into $C = 2$ registers with $n_0 = 4$, $n_1 = 2$, and let $t = 2$, $L = 4$ and $D = 2$. Assume that between two adjacent updates, an error occurs in each register. Let the WOM codes of r_0 and r_1 correct 2 and 1 errors, respectively. Let the rewriting graph $\mathcal{G}$ whose vertex and edge sets be defined as $V = \{v_0, v_1, v_2, v_3\}$, $E = \left\{ v_0 \overset{(0)}{\underset{(0)}{\rightleftarrows}} v_1, v_1 \overset{(1)}{\underset{(0)}{\rightleftarrows}} v_2, v_2 \overset{(1)}{\underset{(0)}{\rightleftarrows}} v_3, v_3 \overset{(1)}{\underset{(1)}{\rightleftarrows}} v_0 \right\}$, where the vertex v_i representing the symbol i and having two outgoing edges locally labeled with (0) and (1). Let the sequence of messages be $(0,3,2,1)$, which corresponds to the path $v_0 \overset{(1)}{\to} v_3 \overset{(0)}{\to} v_2 \overset{(0)}{\to} v_1$ in the graph. Assume that the changes on the states of r_0 and r_1 during the updates are the ones shown in the table below. Here j^- and j^+ denote the moments immediately before and after the j-th update, respectively. A bit marked with underlines indicates an error. Note that at the moment $j = 2$, although performing the update does not require recovery of the messages written at moments $j = 0$ and $j = 1$, those messages can still be recovered until the moment $j = 2^-$ if needed.*

6.2.3 Analysis of the Correctness of the Construction

To see the correctness of the coding scheme, we use induction. (Here we assume the number of cells goes to infinity.) Let us assume that the first j messages have been stored

Table 6.1: An example of the first construction.

j	r_0	r_1	**Comments**
0^-	$(0,0,0,0)$	$(0,0)$	Initialization
0^+	$(0,1,0,0)$	$(0,0)$	Wrote data “0” in r_0
1^-	$(0,1,\underline{1},0)$	$(0,0)$	An error occurs in r_0
1^+	$(0,1,\underline{1},0)$	$(1,0)$	Decoded r_0, wrote “(1)” in r_1
2^-	$(0,\underline{0},\underline{1},0)$	$(\underline{0},0)$	Errors occur in r_0 and r_1
2^+	$(1,0,1,0)$	$(\underline{0},0)$	Rewrote r_0 to store “2”
3^-	$(1,0,\underline{0},0)$	$(\underline{0},\underline{1})$	Errors occur in r_0 and r_1.
3^+	$(1,0,\underline{0},0)$	$(0,1)$	Decoded r_0, wrote “(0)” in r_1

successfully, and we show that M_{j-1} can be recovered reliably at any time between the $(j-1)$-th and the j-th update, and the j-th message can be stored successfully. Let the index of the register be written as $l = j \bmod C$. If $l = 0$, we are at the first write of a new cycle, and do not need to recover M_{j-1} to store M_j; if $l > 0$, we perform the update by storing the difference Δ_j between M_{j-1} and M_j in r_l. To do so, we first recover the value of M_{j-1} by decoding the registers r_0, r_1, $\cdots$, r_{l-1} which have respectively received errors from the channels $\text{BSC}(p^{*l})$, $\text{BSC}(p^{*(l-1)})$, $\cdots$, $\text{BSC}(p)$ at the time of the decoding. As their WOM codes respectively correct errors from $\text{BSC}(p^{*C})$, $\text{BSC}(p^{*(C-1)})$, $\cdots$, $\text{BSC}(p^{*(C-l+1)})$ which are degraded versions of their current noise channels, these registers can be decoded, outputting the messages written by the last l updates (which include $\hat{M}_{j-l}$ stored in r_0, and the labels $\hat{\Delta}_{j-l+1}, \hat{\Delta}_{j-l+2}, \cdots, \hat{\Delta}_{j-1}$ from r_1, $\cdots$, r_{l-1}). Given $\mathcal{G}(V, E)$ we can determine the value of $\hat{M}_{j-1}$, and further compute the label Δ_j of the edge from $\pi(M_{j-1})$ to $\pi(M_j)$. By storing the label Δ_j into r_l, the j-th update succeeds.

6.2.4 Code Analysis

We analyze the code performance for Construction 4. Let $L = |V|$. For $j \in [t]$, let $R_{i,j} > 0$ be the achievable instantaneous rate of the j-th write of the EC-WOM code in

r_i. As each register uses a constant-rate WOM code (here register r_0 stores $\log_2 L$ bits per write, and the other registers each stores $\log_2 D$ bits per write), for $i \in [C-1]$ we have

$$n_0 = \frac{\log_2 L}{\min_{j\in[t]} R_{0,j}}, n_i = \frac{\log_2 D}{\min_{j\in[t]} R_{i,j}}. \tag{6.2}$$

Substituting Eq. (6.2) in Eq. (6.1) gives the rate of the code

$$R = \frac{t\cdot C}{\frac{1}{\min_{j\in[t]} R_{0,j}} + \frac{\log_2 D}{\log_2 L}\sum_{i=1}^{C-1}\frac{1}{\min_{j\in[t]} R_{i,j}}}.$$

Note that the EC-WOM in Construction 4 is general. To be specific, we can use the polar EC-WOM code in Chapter 5 for each register, and derived the bounds on R. We first revisit some results in Chapter 5 that are needed to derive the bounds to the instantaneous rates for the polar EC-WOM code.

Let the WOM channel used for performing the j-th write/encoding of the polar EC-WOM be $\mathrm{WOM}(\alpha_{j-1}, \epsilon_j)$ with the parameters α_{j-1} and ϵ_j, and let the channel of noise in cell states between two adjacent writes be $\mathrm{BSC}(p_e)$. Let $F_{\mathrm{WOM}(\alpha_{j-1},\epsilon_j)} \subseteq [N]$ be the frozen set of the polar code constructed for $\mathrm{WOM}(\alpha_{j-1}, \epsilon_j)$, and let $F_{\mathrm{BSC}(p_e)} \subseteq [N]$ be the frozen set of the code constructed for $\mathrm{BSC}(p_e)$. When $N \to \infty$, let $x_j \triangleq |F_{\mathrm{WOM}(\alpha_{j-1},\epsilon_j)} \cap F_{\mathrm{BSC}(p_e)}|/|F_{\mathrm{BSC}(p_e)}| \leqslant 1$. For $j \in [t]$, the number of bits written in the j-th rewrite is $\mathcal{M}_j = |F_{\mathrm{WOM}(\alpha_{j-1},\epsilon_j)}| - |F_{\mathrm{WOM}(\alpha_{j-1},\epsilon_j)} \cap F_{\mathrm{BSC}(p_e)}| = N\alpha_{j-1}\,\mathrm{H}(\epsilon_j) - x_j|F_{\mathrm{BSC}(p_e)}| = N(\alpha_{j-1}\,\mathrm{H}(\epsilon_j) - x_j\,\mathrm{H}(p_e))$ and the number of additional cells we use to store the bits in $F_{\mathrm{BSC}(p)} - F_{\mathrm{WOM}(\alpha_{j-1},\epsilon_j)}$ is $N_{\mathrm{additional},j} = \frac{N\,\mathrm{H}(p_e)(1-x_j)}{1-\mathrm{H}(p_e)}$. Therefore, we get the instantaneous rate for the j-th write

$$R_j \triangleq \frac{\mathcal{M}_j}{N + \sum_{k=1}^{t} N_{\mathrm{additional},j}} = \frac{\alpha_{j-1}\,\mathrm{H}(\epsilon_j) - \mathrm{H}(p_e)x_j}{1 + \frac{\mathrm{H}(p_e)}{1-\mathrm{H}(p_e)}\sum_{k=1}^{t}(1-x_k)}.$$

Lemma 21. *Let $0 < p_e \leqslant \alpha_{j-1}\epsilon_j$. Then $R_j \in [R_j^-, R_j^+]$, where*

$$R_j^- = \frac{\alpha_{j-1}\,\mathrm{H}(\epsilon_j) - \mathrm{H}(p_e)}{1 + \frac{\mathrm{H}(p_e)}{1-\mathrm{H}(p_e)}\sum_{k=1}^{t}(1-\gamma_k)}, \tag{6.3}$$

$$R_j^+ = \alpha_{j-1}\,\mathrm{H}(\epsilon_j) - \mathrm{H}(p_e)\gamma_j. \tag{6.4}$$

The results above can be directly applied to the codes in Construction 4. For $i \in \{0, 1, \cdots, C-1\}$, let $0 < p^{*(C-i)} \leqslant \alpha_{i,j-1}\epsilon_{i,j}$, then $R_{i,j} \in [R_{i,j}^-, R_{i,j}^+]$ where $R_{i,j}^-$ and $R_{i,j}^+$ are computed with the right hand sides of Eq. (6.3) and (6.4) by replacing α_{j-1}, ϵ_j and p_e with $\alpha_{i,j-1}$, ϵ_j and $p^{*(C-i)}$.

Theorem 22. *For $i \in \{0, 1, \cdots, C-1\}$, $j \in [t]$, let $0 < p^{*(C-i)} \leqslant \alpha_{i,j-1}\epsilon_{i,j}$. Then $R \in \{R^-, R^+\}$ where*

$$R^- = \frac{t \cdot C}{\frac{1}{\min_{j\in[t]} R_{0,j}^-} + \frac{\log_2 D}{\log_2 L}\sum_{i=1}^{C-1}\frac{1}{\min_{j\in[t]} R_{i,j}^-}}.$$

and the upper bound R^+ can be computed by replacing $R_{0,j}^-$ and $R_{i,j}^-$ in the above equation with $R_{0,j}^+$ and $R_{i,j}^+$, respectively.

Figure 6.1 shows some numerical results for the bounds of our code, where for all i, j we let $\epsilon_{i,j} = 1/(2 + t - j)$. To see its good performance, we compare the bounds of our scheme to those of the basic scheme, which is simply a Ct-write polar EC-WOM code correcting errors from $\mathrm{BSC}(p)$. In each rewrite, the basic scheme stores each updated message using rewriting. The results suggest our code performs significantly better than the basic scheme. (Note that the WOM codes considered in this work are constant rate codes. Given such codes, the bounds in Figure 6.1 decreases when t becomes sufficiently large due to the drop in the instantaneous rates.)

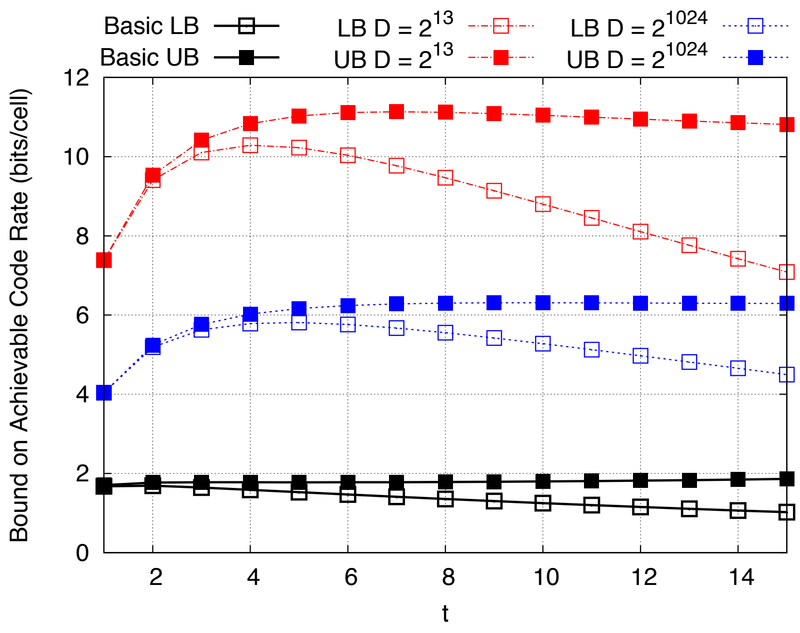


Figure 6.1: The lower and upper bounds (marked by LB and UB) on the achievable code rates for different t and D. Here $\log_2 L = 2^{13}$, $C = 8$ and $p = 10^{-3}$.

6.3 A More Generalized Coding Scheme

We now discuss a more generalized coding scheme. In this scheme, the trajectory codes not only use registers to store the changes in the messages, but can also store part of the errors found in previous registers. When the error probability of the channel is small, only a small number of additional cells are needed to store such error information. We focus on a specific construction in the following.

6.3.1 Code Construction

Let the error-free cell states of register r_i (immediately after it is written) be $\boldsymbol{c}_i^0 \in \{0,1\}^{n_i}$. Let the cell states immediately before each of the next C updates of messages be $\boldsymbol{c}_i^1, \boldsymbol{c}_i^2, \cdots, \boldsymbol{c}_i^C$. According to the error model in Section 6.2, the error vector $\boldsymbol{c}_i^k \oplus \boldsymbol{c}_i^{k+1}$

contains the errors introduced by $\mathrm{BSC}(p)$. When $n_i \to \infty$, the vector $\boldsymbol{c}_i^k \oplus \boldsymbol{c}_i^{k+1}$ can be compressed into $n_i \, \mathrm{H}(p)$ bits using lossless source coding. The encoder and the decoder for the j-th update in the new construction are defined below.

Construction 5. *For $i \in \{0, 1, \cdots, C-1\}$, let register r_i use a t-write EC-WOM code, correcting the errors from $\mathrm{BSC}(p)$. For $j = 0, 1, \cdots, Ct-1$, we have*

Encoder $\mathbf{E}_j(\boldsymbol{s}_j, M_j, \mathcal{G}(V, E)) = \boldsymbol{s'}_j$

If $l = 0$, rewrite r_0 with M_j. Otherwise, do;

(1) Recover message $\hat{M}_{j-1} = \mathbf{D}_{j-1}(\boldsymbol{s}_j, \mathcal{G}(V, E))$.

(2) Compute the label Δ_j s.t. $(\pi(\hat{M}_{j-1}) \xrightarrow{\Delta_j} \pi(M_j)) \in E$.

(3) Rewrite register r_j to store Δ_j and the compressed version of the error vectors $\boldsymbol{c}_{l-1}^0 \oplus \boldsymbol{c}_{l-1}^1$, $\boldsymbol{c}_{l-2}^1 \oplus \boldsymbol{c}_{l-2}^2, \cdots, \boldsymbol{c}_0^{l-1} \oplus \boldsymbol{c}_0^l$.

Decoder $\mathbf{D}_j(\boldsymbol{s'}_j, \mathcal{G}(V, E)) = \hat{M}_j$

(1) For k from 0 to l, let the state of register r_{l-k} be $\boldsymbol{c}_{l-k}^{k+1}$. Using it and the error vectors obtained previously from decoding $r_{l-k+1}, r_{l-k+2}, \cdots, r_l$, we get $\boldsymbol{c}_{l-k}^{k+1} \oplus \sum_{x=0}^{k-1}(\boldsymbol{c}_{l-k}^x \oplus \boldsymbol{c}_{l-k}^{x+1}) = \boldsymbol{c}_{l-k}^0 \oplus (\boldsymbol{c}_{l-k}^k \oplus \boldsymbol{c}_{l-k}^{k+1})$. (Note that when $k = 0$, the above equals $\boldsymbol{c}_l^1$.) Decode the right hand side of the above equation, and obtain the recorded error vectors about the first $(l-k)$ registers—$\boldsymbol{c}_0^{l-k} \oplus \boldsymbol{c}_0^{l+1-k}$, $\boldsymbol{c}_1^{l-k-1} \oplus \boldsymbol{c}_1^{l-k}, \cdots, \boldsymbol{c}_{l-k}^0 \oplus \boldsymbol{c}_{l-k}^1$—the estimated message $\hat{\mathcal{M}}_{j-l}$ (when $k = l$) or the estimated edge label $\hat{\Delta}_{j-k}$ (when $k < l$).

(2) We now compute $\hat{M}_j$; we traverse the graph $\mathcal{G}(V, E)$ along the path marked by the labels $\hat{\Delta}_{j-l+1}, \hat{\Delta}_{j-l+2}, \cdots, \hat{\Delta}_j$, which leads to vertex $\hat{v}_j$. Output $\hat{M}_j = \pi^{-1}(\hat{v}_j)$.

Example 23. *Let $n = 10$, $t \geqslant 1$, $C = 3$, $L = 4$ and $D = 2$. Assume $n_0 = 3$, $n_1 = 3$, $n_2 = 4$, and that the WOM code of each register corrects 1 error. Assume that between two adjacent updates, an error occurs in each register. We assume the same rewriting graph as in Example 20, and let $(0, 3, 2)$ be the first three messages to be stored. We only illustrate the update for the message "2" due to space limitation. Assume the changes in the cell states during the updates are as in the table below. At time 2^-, errors occur in r_0 and r_1. To perform the update, we first decode r_1, and obtain the label "(1)" and the decompressed error vector $\boldsymbol{c}_0^0 \oplus \boldsymbol{c}_0^1 = (0, 0, 1)$ for r_0. Given the error vector and the current state $\boldsymbol{c}_0^2$, compute $\boldsymbol{c}_0^2 \oplus (\boldsymbol{c}_0^0 \oplus \boldsymbol{c}_0^1) = (0, 0, 0)$ where the middle bit still contains error. Decoding $\boldsymbol{c}_0^2 \oplus (\boldsymbol{c}_0^0 \oplus \boldsymbol{c}_0^1)$ gives the message "0". Given the new message "$2''$ and the recovered label "(1)" and the message "$0''$ in r_0, the label "(0)" is determined and stored by writing "(0)" in r_2, which completes the update.*

Table 6.2: An example of the second construction.

j	r_0	r_1	r_2
0^-	$(0,0,0)$	$(0,0,0)$	$(0,0,0,0)$
0^+	$(0,1,0)$	$(0,0,0)$	$(0,0,0,0)$
1^-	$(0,1,\underline{1})$	$(0,0,0)$	$(0,0,0,0)$
1^+	$(0,1,\underline{1})$	$(1,0,0)$	$(0,0,0,0)$
2^-	$(0,\underline{0},\underline{1})$	$(1,\underline{1},0)$	$(0,0,0,0)$
2^+	$(0,\underline{0},\underline{1})$	$(1,\underline{1},0)$	$(1,0,1,0)$

6.3.2 Analysis of the Correctness of the Code

The correctness of Construction 5 can be shown by induction. (We again assume the number of cells in each register goes to infinity.) Assume the first j messages have been stored successfully, and we elaborate on the j-th update with $l > 0$. To perform the update, we need to recover the message M_{j-1} so that the label of the edge from M_{j-1} to M_j can be computed and stored in r_l. We first decode r_{l-1} with state $\boldsymbol{c}_{l-1}^1$ which received the errors from $\mathrm{BSC}(p)$. Since each register tolerates errors from $\mathrm{BSC}(p)$, r_{l-1} can be decoded to obtain the edge label Δ_{j-1} (that specifies the edge connecting M_{j-2} to M_{j-1}) as well as the error vectors $\boldsymbol{c}_{l-2}^0 \oplus \boldsymbol{c}_{l-2}^1$, $\boldsymbol{c}_{l-3}^1 \oplus \boldsymbol{c}_{l-3}^2$, $\cdots$, $\boldsymbol{c}_0^{l-2} \oplus \boldsymbol{c}_0^{l-1}$ with each error vector being for one of the first $l-1$ registers. Next, we decode r_{l-2} with state $\boldsymbol{c}_{l-2}^2$. To do so, we first use the error vector obtained previously on r_{l-2} to correct part of the errors by computing $\boldsymbol{c}_{l-2}^2 \oplus (\boldsymbol{c}_{l-2}^0 \oplus \boldsymbol{c}_{l-2}^1)$. The remaining errors can be equivalently seen as coming from $\mathrm{BSC}(p)$, and are thus correctable. Decoding them gives the edge label Δ_{j-2} as well as the error vectors regarding the previous registers. We continue the joint decoding in the same fashion towards r_0. Thanks to the error vectors from the previous decoding, each register needs to correct errors from $\mathrm{BSC}(p)$ (instead of $\mathrm{BSC}(p^{*(C-i)})$ for $i = 0, 1, \cdots, C-1$). After r_0 is decoded, we obtain the message M_{j-l} and the labels $\Delta_{j-l+1}, \Delta_{j-l+2}, \cdots, \Delta_{l-1}$. By traversing $\mathcal{G}(V, E)$ along the path marked by the labels, we recover $\hat{M}_{j-1}$. The label Δ_j is then determined and written into r_l.

6.3.3 Code Analysis

We analyze the code performance of Construction 5. The analysis is different from Construction 4 mainly for two reasons. The EC-WOM code of each register for the codes of this section corrects the errors from $\mathrm{BSC}(p)$ while each WOM code tolerates different amount of noise in the previous construction. Moreover, since each register (besides r_0) stores both error vectors as well as an edge label, the value of n_i also depends on $n_0, n_1, \cdots, n_{i-1}$.

We first derive n_i for each r_i. As r_0 is used in the same way as the previous codes, and r_i stores i error vectors and one edge label in each write, we have $n_0 = \log_2 L / \min_{j\in[t]} R_{0,j}$ and $n_i = (\log_2 D + \mathrm{H}(p) \sum_{k=0}^{i-1} n_k) / \min_{j\in[t]} R_{i,j}$ for $i \in [C–1]$. Here the term $\mathrm{H}(p) \sum_{k=0}^{i-1} n_k$

is the length of the compressed error vectors $\boldsymbol{c}_{i-1}^{0} \oplus \boldsymbol{c}_{i-1}^{1}$, $\boldsymbol{c}_{i-2}^{1} \oplus \boldsymbol{c}_{i-2}^{2}$, $\cdots$, $\boldsymbol{c}_{0}^{i-1} \oplus \boldsymbol{c}_{0}^{i}$. In practice, each register can choose to use the WOM code with the same parameters to simplify the implementation. In such cases, $(n_1, n_2, \cdots, n_{C-1})$ form a geometric sequence.

Proposition 24. *For $i \in \{1, 2, \cdots, C-1\}$, let $\min_{j\in[t]} R_{i,j}$ be some constant A. Then we have $n_i = (n_0 \, \mathrm{H}(p) + \log_2 D)(A + \mathrm{H}(p))^{i-1}/A^i$.*

Therefore, the rate of the code in this section can be computed using Eq. (1). To derive the bounds for Construction 5, we apply the same techniques used in Section 6.2. Assume each WOM code in is a polar EC-WOM code which corrects errors from $\mathrm{BSC}(p)$. By applying Lemma 21, we show the bounds to the instantaneous rates $R_{i,j}$ in the next lemma.

Lemma 25. *For $i \in \{0, 1, \cdots, C-1\}$ and $j \in [t]$, let $0 < p \leqslant \alpha_{i,j-1}\epsilon_{i,j}$. Then we have $R_{i,j} \in [R_{i,j}^{-}, R_{i,j}^{+}]$, where $R_{i,j}^{-} = [\alpha_{i,j-1} \mathrm{H}(\epsilon_{i,j}) - \mathrm{H}(p)]/\left[1 + \frac{\mathrm{H}(p)}{1-\mathrm{H}(p)} \sum_{j=1}^{t}(1-\gamma_{i,j})\right]$ and $R_{i,j}^{+} = \alpha_{i,j-1} \mathrm{H}(\epsilon_{i,j}) - \mathrm{H}(p)\gamma_{i,j}$.*

Theorem 26. *For all i and j, let $0 < p \leqslant \alpha_{i,j-1}\epsilon_{i,j}$. Then we have $R \in [R^-, R^+]$, where $R^- = Ct \log_2 L/\left[\frac{\log_2 L}{\min_{j\in[t]} R_{0,j}^{-}} + \sum_{i\in[C-1]} \frac{\log_2 D + \mathrm{H}(p)\sum_{k=0}^{i-1} n_k}{\min_{j\in[t]} R_{i,j}^{-}}\right]$, and R^+ can be computed by replacing $R^-(0,j)$ and $R^-(i,j)$ in R^- above with $R^+(0,j)$ and $R^+(i,j)$, respectively.*

Figure 6.2 shows the numerical results that compare the bounds of Construction 4 and Construction 5 on parameters that are common for flash memories (e.g. message length > 1000 bits). The bounds for the codes in this section are tighter than those of the previous construction. When t is sufficiently large, all bounds will decrease due to the decrease of the minimum instantaneous rates. However, the bounds of the codes in this section decrease more slowly. This is because in the first construction, the WOM code of r_i needs to tolerate the errors from $\mathrm{BSC}(p^{*(C-i)})$. Its error rates become much higher than what the codes in this section needs to tolerate (which is $\mathrm{BSC}(p)$) when C becomes large.

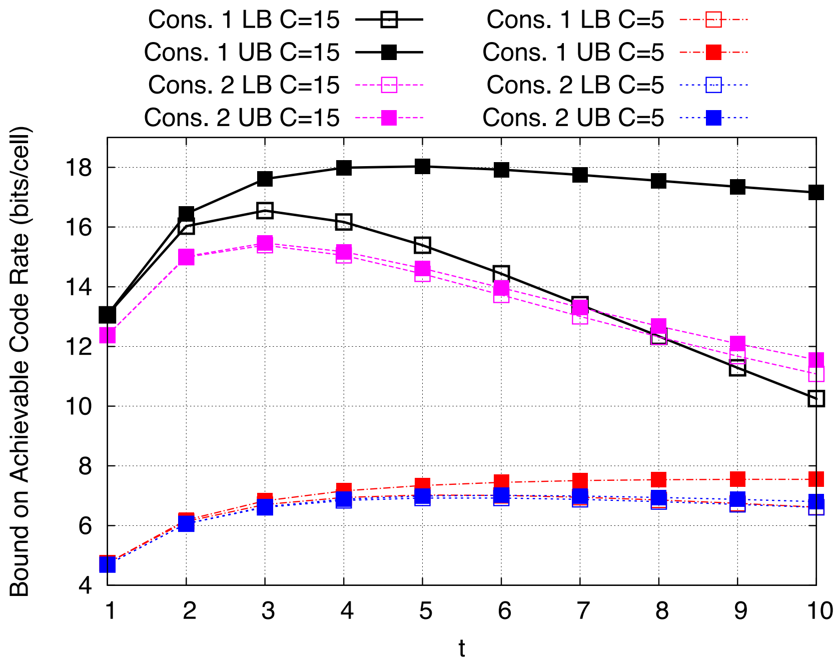


Figure 6.2: The bounds to the achievable rates of the two constructions on different t and C. Here $\log_2 L = 2^{13}$, $p = 10^{-3}$, and $\epsilon_{i,j} = \frac{1}{2+t-j}$.

Therefore, the minimum instantaneous rates of the WOM codes in the previous scheme decrease faster when t increases than those of the codes in this section do.

7. WOM CODES THAT CORRECT WRITE ERRORS

An extension of the concept of polarization [2] to asymmetric settings was proposed in [28] in the context of channel and lossy source coding. In this chapter we apply this extension to the problem of noisy WOM coding, an asymmetric channel coding problem with state information at the encoder.

The main contribution of this work is the derivation of a new nested polar coding scheme for the noisy (asymmetric) WOM channel with an asymptotically-optimal rate and polynomial complexity. We extend the result of [10] to the noisy WOM case, where in contrast to [10], the amount of required shared randomness is significantly reduced, which increases the practical applicability of the scheme. In particular, if the cell states are assumed to be i.i.d., the requirement for shared randomness is removed completely. The encoding and decoding complexity of the proposed scheme is given as $O(n \log n)$ under a decoding error probability of $2^{-\Omega(n^{1/2-\delta})}$ for a block length n and any $\delta > 0$. As a byproduct, we identify how the proposed technique can be used to provide a nested coding solution for other asymmetric side-information based problems.

7.1 WOM Channel Model

A WOM is composed of a block of n cells, where each cell has a state from some finite alphabet. In this work we assume that the alphabet is $\{0, 1\}$. The main property of the WOM is that a cell at state 0 can change its state to 1, but once the cell state is 1, it cannot be changed anymore. The state of the cells is known to a user that wishes to store information on the memory. If some of the cells are in state 1, a code is required for the reliable storage of information, since not every sequence of n bits can be stored directly.

We study a *stochastic* i.i.d. WOM model. Let S be a Bernoulli random variable that corresponds to the state of a cell. Since the cells are i.i.d., we define the distribution of a cell state by

$$P(S = 1) = \beta, \tag{7.1}$$

for some $\beta \in [0, 1]$. Although the standard model of WOM used for flash memories does not consider independently distributed cell states, the i.i.d. model can be applied to flash memories using a method described in [10].

The focus of this work is on memories with *write errors*, where the corresponding bit channel is shown in Figure 7.1. Here X and Y are Bernoulli random variables that correspond to the input and output of a single bit in the memory. If the cell state S is 1, the output Y will be 1 as well, regardless of the value of input X, corresponding to a stuck bit. Otherwise, if the cell state S is 0, we assume that the memory exhibits a symmetric writing error with crossover probability α. This behavior is summarized by

$$P_{Y|XS}(1|x, s) = \begin{cases} \alpha & \text{if } (x, s) = (0, 0) \\ 1 - \alpha & \text{if } (x, s) = (1, 0) \\ 1 & \text{if } s = 1. \end{cases} \tag{7.2}$$

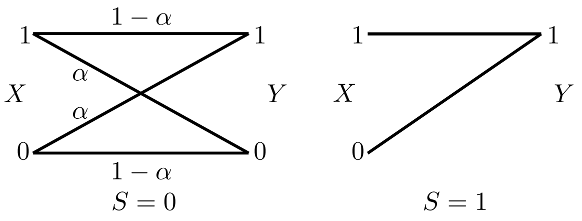


Figure 7.1: A binary WOM with write errors.

Finally, in our model of study, we limit the number of cells that the encoder attempts to change from 0 to 1 in the memory. This limitation lends itself naturally to the application of WOM to settings in which multiple writes are considered. We express this limitation by a parameter ϵ that bounds the expectation of the input X given that $S = 0$ as follows

$$E(X|S = 0) \leqslant \epsilon. \tag{7.3}$$

The capacity of the WOM model is given in the following theorem.

Theorem 27. *[27, Theorem 4] The capacity of the memory described by (7.1), (7.2), and (7.3) is $\mathcal{C} = (1 - \beta)[h(\epsilon * \alpha) - h(\alpha)]$, where $\epsilon * \alpha \equiv \epsilon(1 - \alpha) + (1 - \epsilon)\alpha$.*

The nested polar coding scheme of this work achieves the capacity of Theorem 27. The presentation of the scheme requires some notation from polar coding.

7.2 Polar Coding Notation

For a positive integer n, let $[n] \equiv \{1, 2, \dots, n\}$. Let $X_1^n = (X_1, X_2, \dots, X_n)$, $Y_1^n = (Y_1, Y_2, \dots, Y_n)$ and $S_1^n = (S_1, S_2, \dots, S_n)$ be i.i.d. copies of X, Y and S, respectively. For integers $i < j$, let X_i^j represent the subvector $(X_i, X_{i+1}, \dots, X_j)$ and for a set $\mathcal{A} \subset [n]$ let $X_{\mathcal{A}}$ represent the subvector $\{X_i\}_{i \in \mathcal{A}}$.

Let n be a power of 2. Define a matrix $G_n = G^{\otimes \log_2 n}$ where $G = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ and $\otimes$ denotes the Kronecker power. A subset of the rows of G_n serves as a generator matrix in channel polar coding schemes. Let U_1^n be the product $U_1^n = X_1^n G_n^{-1} = X_1^n G_n$. A subset of the coordinates of the vector U_1^n will contain the message to be stored in the memory.

Our analysis uses the Bhattacharyya parameters. For a conditional distribution $P_{Y|X}$, where X is a Bernoulli random variable, the Bhattacharyya parameter is defined by

$$Z_B(P_{Y|X}) \triangleq \sum_y \sqrt{P_{Y|X}(y|0)P_{Y|X}(y|1)}.$$

The Bhattacharyya parameter serves as an upper bound on the decoding error probability in polar codes for symmetric settings. Similarly, define the conditional Bhattacharyya parameter as

$$Z(X|Y) \triangleq 2\sum_{y}\sqrt{P_{X,Y}(0,y)P_{X,Y}(1,y)}.$$

The conditional Bhattacharyya parameter serves as a decoding error probability bound in polar codes for asymmetric settings. Note that the Bhattacharyya parameter and the conditional Bhattacharyya parameter are equal if X is distributed uniformly. The conditional Bhattacharyya parameter $Z(X|Y)$ is related to the conditional entropy $H(X|Y)$ according to the following proposition.

Proposition 28. *([3, Proposition 2])*

$$(Z(X|Y))^2 \leqslant H(X|Y), \tag{7.4}$$

$$H(X|Y) \leqslant \log_2(1+Z(X|Y)) \leqslant Z(X|Y). \tag{7.5}$$

We now turn to describe the proposed coding scheme.

7.3 Coding Scheme

We start by presenting a rough overview of our coding technique followed by a formal presentation. The achievability of Theorem 27 is shown by random coding. The distribution by which the random codes are drawn in this achievability proof is called the capacity-achieving distribution, and it is used in our scheme. We denote this distribution by $P_{X|S}$. According to the proof of [27, Theorem 4], we have

$$P_{X|S}(1|0) = \epsilon, \qquad P_{X|S}(1|1) = \frac{\epsilon(1-\alpha)}{\epsilon * \alpha}. \tag{7.6}$$

Our construction is based on a combination of two methods: *asymmetric* polar cod-

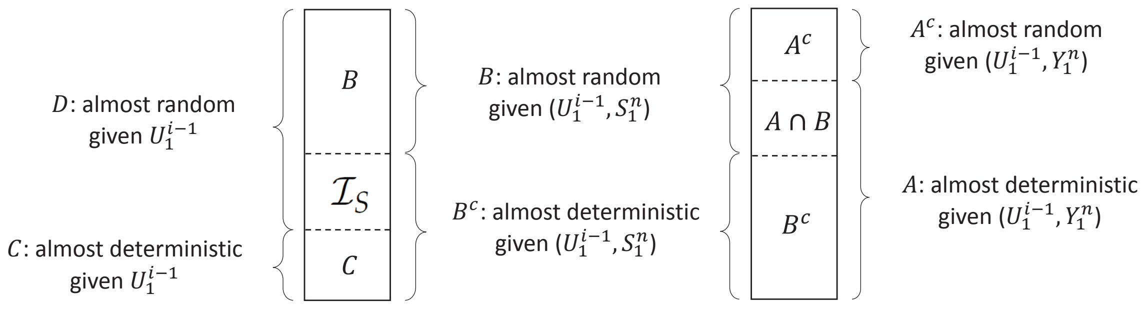

D: almost random given U_1^{i-1}
C: almost deterministic given U_1^{i-1}
B
$\mathcal{I}_S$
C
B: almost random given (U_1^{i-1}, S_1^n)
B^c: almost deterministic given (U_1^{i-1}, S_1^n)
A^c
$A \cap B$
B^c
A^c: almost random given (U_1^{i-1}, Y_1^n)
A: almost deterministic given (U_1^{i-1}, Y_1^n)

Figure 7.2: Different polarizations of $U_1^n = X_1^n G_n$.

ing [28] and *nested* polar coding [45]. In nested polar coding for Gelfand-Pinsker-type problems, the encoder first *compresses* the state using a lossy source code, and then transmits the compressed state together with the source message using a channel code. To achieve capacity, both the lossy source coding and the channel coding follow the capacity-achieving conditional distribution of Equation (7.6).

Let s_1^n, u_1^n, x_1^n and y_1^n be the realizations of the random variables S_1^n, U_1^n, X_1^n and Y_1^n, respectively. In a channel polar coding scheme, a vector u_1^n is used for representing the source message and a frozen vector. The channel input is the codeword $x_1^n = u_1^n G_n$. The coding scheme is *polarizes* the conditional entropies $H(U_i|U_1^{i-1}, Y_1^n)$ for the different coordinates i of the vector U_1^n. In a nested polar coding scheme, the vector U_1^n is also polarized with respect to the conditional entropies $H(U_i|U_1^{i-1}, S_1^n)$. In our case the conditional entropies $H(U_i|U_1^{i-1}, S_1^n)$ are defined according to the capacity-achieving conditional distribution of Equation (7.6). In addition, in asymmetric polar coding, the vector U_1^n is also polarized with respect to the conditional entropies $H(U_i|U_1^{i-1})$. We take advantage of these three different polarizations of the vector U_1^n in our coding scheme. It is useful to define the polarized sets according to the conditional Bhattacharyya parameter, as follows:

$$
\begin{aligned}
A &\equiv \{i : Z(U_i|U_1^{i-1}, Y_1^n) \leqslant 2^{-n^{1/2-\delta}}\},\\
B &\equiv \{i : Z(U_i|U_1^{i-1}, S_1^n) \geqslant 1 - 2^{-n^{1/2-\delta}}\},\\
C &\equiv \{i : Z(U_i|U_1^{i-1}) \leqslant 2^{-n^{1/2-\delta}}\},\\
D &\equiv \{i : Z(U_i|U_1^{i-1}) \geqslant 1 - 2^{-n^{1/2-\delta}}\},
\end{aligned}
$$

for some small $\delta > 0$. The sets A, B, C and D refer to sets of coordinated of the vector u_1^n. Also define the sets A^c, B^c, C^c and D^c to be the complements of A, B, C and D with respect to $[n]$. Intuitively, set A contains the coordinates of u_1^n that can be decoded reliably given y_1^n. Set B contains the coordinates that have small relation to s_1^n. Set C is composed of the coordinates that do not contain much information about x_1^n, and set D is composed of the coordinates that contain almost all the information about x_1^n. Note that those sets are not disjoint. An illustration of the relation between those sets is presented in Figure 7.2.

The proposed coding scheme takes advantage of the polarized sets as follows. First, the encoder performs a lossy compression of the state s_1^n by the method of [28]. The compression is performed according to an information set $\mathcal{I}_S$, defined to be the set of coordinates that are *not* in the union $B \cup C$. The complement of the information set with respect to $[n]$ is denoted as $\mathcal{I}_S^c = B \cup C$. Each bit u_i for $i \in \mathcal{I}_S$ is set randomly to a value u with probability $P_{U_i|U_1^{i-1},S_1^n}(u|u_1^{i-1}, s_1^n)$. The coordinates in the set B are distributed almost uniformly given (U_1^{i-1}, S_1^n), and therefore they do not affect the joint distribution (X_1^n, S_1^n). For that reason we can place the source message in those coordinates. Since we also need the message to be decoded reliably given Y_1^n, we restrict it to the coordinates in the set A, which are almost deterministic given (U_1^{i-1}, Y_1^n). Taking both restrictions into account, we set the coordinated of u_1^n in the intersection $A \cap B$ to be equal to the source message.

The rest of the coordinates of u_1^n are set by the encoder according to a set of Boolean functions, as in [28]. Since this set of functions describes the code, it is known to both the encoder and the decoder. For a positive integer i, a Boolean function is denoted by $\lambda_i : \{0,1\}^{i-1} \to \{0,1\}$. Remember that the coordinates in the set $\mathcal{I}_S$ are determined according to the state s_1^n, and that the coordinates in $A \cap B$ are determined by the information message. The rest of the coordinates of u_1^n will be determined according to a set of Boolean functions $\lambda_{\mathcal{I}_S^c \setminus (A\cap B)} = \{\lambda_i\}_{\mathcal{I}_S^c \setminus (A\cap B)}$. The encoder sets a bit u_i according to the function $\lambda_i(u_1^{i-1})$.

Finally, remember that the set A contains the reliable coordinates of u_1^n given y_1^n. Therefore, the decoder can guess the coordinates in A, and will estimate the rest of the coordinates according to $\lambda_{\mathcal{I}_S^c \setminus A}$. A coordinate i in $\mathcal{I}_S^c$ can be decoded this way, since if $i \in A$, u_i can be estimated reliably, and if $i \in \mathcal{I}_S^c \setminus A$, then u_i can be recovered by λ_i. However, a coordinate $i \in \mathcal{I}_S$ can be decoded reliably only if $i \in A$, since we do not use Boolean functions for the set $\mathcal{I}_S$. Therefore, a reliable decoding requires $\mathcal{I}_S$ to be a subset of A. While we do not know if this fact always holds, we can show that the set difference $\mathcal{I}_S \setminus A$ is very small. Therefore, our strategy is to store the vector $u_{\mathcal{I}_S \setminus A}$ in additional cells. Since we show that the fraction $|\mathcal{I}_S \setminus A|/n$ approaches zero for large n, this strategy will not affect the asymptotic rate of the scheme or the expected weight of the codewords.

The additional cells should be coded as well to ensure their reliability. In this work we assume that the additional cells are protected by an asymmetric channel polar code, without utilizing the state information at the encoder. We do not discuss the details of the coding of the additional cells. However, we note that since the block length of the additional cells is much smaller than the main block length, the probability of decoding error at the additional cells is higher. In particular, the probability of decoding error at the additional cells is $2^{-\Omega(n^{1/2-\delta})}$ for any $\delta' > \delta$, while the decoding error probability in the main block of cells is $O(2^{-n^{1/2-\delta}})$, as in standard polar codes. We now describe the coding scheme formally.

Construction 6.

Encoding

Input: *a message* $m_1^k \in \{0,1\}^k$ *and a state* $s_1^n \in \{0,1\}^n$.

Output: *a codeword* $x_1^n \in \{0,1\}^n$.

1. *Let* $\boldsymbol{u}_{A\cap B} = m_1^k$*. Then for* i *from* 1 *to* n*, if* $i \in \mathcal{I}_S$*, set*

$$u_i = \begin{cases} 0 & \textit{with probability } P_{U_i|U_1^{i-1},S_1^n}(0|u_1^{i-1}, s_1^n) \\ 1 & \textit{with probability } P_{U_i|U_1^{i-1},S_1^n}(1|u_1^{i-1}, s_1^n), \end{cases}$$

   and if $i \in \mathcal{I}_S^c \setminus (A \cap B)$*, set* $u_i = \lambda_i(u_1^{i-1})$.

2. *The vector* $u_{\mathcal{I}_S\setminus A}$ *is stored in additional cells. Finally, store the codeword* $x_1^n = u_1^n G_n$.

Decoding

Input: *a noisy vector* y_1^n.

Output: *a message estimation* $\hat{m}_1^k$.

Estimate u_1^n *by* $\hat{u}_1^n(y_1^n, \lambda_{\mathcal{I}_S^c\setminus A})$ *as follows:*

1. *Recover the vector* $u_{\mathcal{I}_S\setminus A}$ *from the additional cells, and assigns* $\hat{u}_{\mathcal{I}_S\setminus A} = u_{\mathcal{I}_S\setminus A}$.

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

$$\hat{u}_i = \begin{cases} \arg\max_u P_{U_i|U_1^{i-1},Y_1^n}(u|\hat{u}_1^{i-1}, y_1^n) & \textit{if } i \in A \\ \lambda_i(\hat{u}_1^{i-1}) & \textit{if } i \in \mathcal{I}_S^c \setminus A. \end{cases}$$

3. *Return the estimated message* $\hat{m}_1^k = \hat{u}_{A\cap B}$.

Let n' be the number of additional cells needed to store the vector $u_{\mathcal{I}_S \setminus A}$. The main theorem of this work addresses the properties of Construction 6.

Theorem 29. *Let the message m_1^k be distributed uniformly. Then for any constant $\delta' > \delta > 0$ there exists a set of Boolean functions $\lambda_{\mathcal{I}_S \setminus (A \cap B)}$ for which Construction 6 satisfies the following:*

1. *The rate of the scheme is asymptotically optimal. Formally, $\lim_{n\to\infty} k/(n+n') = \mathcal{C}$.*
2. *The decoding error probability is $2^{-\Omega(n^{1/2-\delta'})}$.*
3. *With the same success probability, for any $\epsilon' > \epsilon > 0$,*

$$\frac{|\{i : s_i = 0 \text{ and } x_i = 1\}|}{|\{i : s_i = 0\}|} \leqslant \epsilon'. \tag{7.7}$$

4. *The encoding and decoding complexities are $O(n \log n)$.*

The complexity claim (claim 2 of Theorem 29) is explained in [28, Section III.B]. In Section 7.4 we prove claim 1 of Theorem 29, and in Section 7.5 we prove claims 2 and 3.

7.4 Optimality of the Code Rate

To prove claim 1 of Theorem 29, we combine two separate claims. First, we show in Subsection 7.4.1 that the fraction of additional cells is negligible, and next, in Subsection 7.4.2, we show that the rate of the main block is asymptotically optimal. Together the two results prove claim 1 of Theorem 29.

7.4.1 Fraction of Additional Cells is Negligible

The vector $u_{\mathcal{I}_S \setminus A}$ is stored on n' additional cells. To ensure a reliable storage, a polar coding scheme requires only a linear amount of redundancy in $|\mathcal{I}_S \setminus A|$. Therefore, to show that the additional cells do not affect the rate and codeword weight of the scheme, it is enough to show that $\lim_{n\to\infty} |\mathcal{I}_S \setminus A|/n = 0$. To show this, we use an idea from the proof of [45, Theorem 15]. Consider the set

$$\begin{aligned}\bar{\mathcal{I}}_S =&\{i : Z(U_i|U_1^{i-1}, S_1^n) \leqslant 2^{-n^{1/2-\delta}} \\ &\text{and } Z(U_i|U_1^{i-1}) \geqslant 1 - 2^{-n^{1/2-\delta}}\}.\end{aligned}$$

By the definition of the sets above, $\bar{\mathcal{I}}_S \subseteq \mathcal{I}_S$. Therefore, by the distributivity of intersection over union, we have

$$|\mathcal{I}_S \setminus A| = |A^c \cap \mathcal{I}_S| = |A^c \cap \mathcal{I}_S \setminus \bar{\mathcal{I}}_S| + |A^c \cap \bar{\mathcal{I}}_S|.$$

To show that $\lim_{n\to\infty} |A^c \cap \mathcal{I}_S|/n = 0$, we start by showing that $\bar{\mathcal{I}}_S \subseteq A$, which implies that $|A^c \cap \bar{\mathcal{I}}_S| = 0$, and therefore that

$$|A^c \cap \mathcal{I}_S| = |A^c \cap (\mathcal{I}_S \setminus \bar{\mathcal{I}}_S)| \leqslant |\mathcal{I}_S \setminus \bar{\mathcal{I}}_S|.$$

From [28, Theorem 1], we have

$$\lim_{n\to\infty} |\mathcal{I}_S \setminus \bar{\mathcal{I}}_S|/n = 0.$$

Therefore, showing that $\bar{\mathcal{I}}_S \subseteq A$ proves that $\lim_{n\to\infty} |A^c \cap \mathcal{I}_S|/n = 0$. To show that $\bar{\mathcal{I}}_S \subseteq A$, it is enough to show that $Z(U_i|U_1^{i-1}, Y_1^n) \leqslant Z(U_i|U_1^{i-1}, S_1^n)$ for all $i \in [n]$, by definitions of the sets $\bar{\mathcal{I}}_S$ and A. To show that $Z(U_i|U_1^{i-1}, Y_1^n) \leqslant Z(U_i|U_1^{i-1}, S_1^n)$ for all $i \in [n]$, we use a sequence of reductions. First, we use the following theorem:

Theorem30. *[28, Theorem 2] Let $\mathcal{Y}$ be the alphabet of Y and let $\tilde{\mathcal{Y}} = \{0, 1\} \times \mathcal{Y}$ and*

$\tilde{Y} = (\tilde{X} \oplus X, Y)$ *where* (X, Y) *is independent of* $\tilde{X}$. *Let* $\tilde{U}_1^n \equiv \tilde{X}_1^n G_n$ *and define*

$$\bar{W}_{Y,i}^{(n)}(\tilde{u}_1^{i-1}, \tilde{y}_1^n | \tilde{u}_i) = P_{\tilde{U}_1^{i-1}, \tilde{Y}_1^n | \tilde{U}_i}(\tilde{u}_1^{i-1}, \tilde{y}_1^n | \tilde{u}_i).$$

Then $Z(U_i | U_1^{i-1}, Y_1^n) = Z_B(\bar{W}_{Y,i}^{(n)})$.

Theorem 30 implies that $Z(U_i|U_1^{i-1}, Y_1^n) \leqslant Z(U_i|U_1^{i-1}, S_1^n)$ if and only if $Z_B(\bar{W}_{Y,i}^{(n)}) \leqslant Z_B(\bar{W}_{S,i}^{(n)})$ for all $i \in [n]$. To show the latter relation, we use the notion of stochastically degraded channels. A discrete memory-less channels (DMC) channel $W_1 : \{0,1\} \to \mathcal{Y}_1$ is stochastically degraded with respect to a DMC channel $W_2 : \{0,1\} \to \mathcal{Y}_2$, denoted as $W_1 \preceq W_2$, if there exists a DMC $W : \mathcal{Y}_2 \to \mathcal{Y}_1$ such that $W_1(y_1|x) = \sum_{y_2 \in \mathcal{Y}_2} W_2(y_2|x) W(y_1|y_2)$. We use the following result on the Bhattacharyya parameters of stochastically degraded channels:

Lemma 31. *[45, Lemma 21] (Degradation of* $W_i^{(n)}$*) Let* $W : \{0,1\} \to \mathcal{Y}_1$ $W' : \{0,1\} \to \mathcal{Y}_2$ *be two discrete memory-less channels (DMC) and let* $W \preceq W'$. *Then for all* i, $W_i^{(n)} \preceq W'^{(n)}_i$ *and* $Z_B(W_i^{(n)}) \geqslant Z_B(W'^{(n)}_i)$.

Lemma 31 reduces the proof of $Z_B(\bar{W}_{Y,i}^{(n)}) \leqslant Z_B(\bar{W}_{S,i}^{(n)})$ for all $i \in [n]$ to showing that $P_{\tilde{S}|\tilde{X}} \preceq P_{\tilde{Y}|\tilde{X}}$. The last degradation relation is proven in the following lemma, which completes the proof that $\lim_{n\to\infty} |\mathcal{I}_S \setminus A|/n = 0$.

Lemma 32.

$$P_{\tilde{S}|\tilde{X}} \preceq P_{\tilde{Y}|\tilde{X}}.$$

Proof: We need to show that there exists a DMC $W : \{0,1\}^2 \to \{0,1\}^2$ such that

$$P_{\tilde{S}|\tilde{X}}(\tilde{s}|\tilde{x}) = \sum_{\tilde{y} \in \{0,1\}^2} P_{\tilde{Y}|\tilde{X}}(\tilde{y}|\tilde{x}) W(\tilde{s}|\tilde{y}). \tag{7.8}$$

To define such channel W, we first claim that

$$P_{Y|X,S}(1|x,0)P_{X|S}(x|0) = (\epsilon * \alpha)P_{X|S}(x|1). \tag{7.9}$$

Equation (7.9) follows directly from Equation (7.6) since

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

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

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

$$\begin{aligned}\frac{P_{X,S}(x,1)}{P_{X,Y}(x,1)} &\overset{\text{(a)}}{=} \frac{P_{X|S}(x|1)P_S(1)}{P_{Y,X|S}(1,x|0)P_S(0) + P_{Y,X|S}(1,x|1)P_S(1)} \\ &\overset{\text{(b)}}{=} P_{X|S}(x|1)\beta/[P_{Y|X,S}(1|x,0)P_{X|S}(x|0)(1-\beta) \\ &\quad + P_{Y|X,S}(1|x,1)P_{X|S}(x|1)\beta] \\ &\overset{\text{(c)}}{=} \frac{P_{X|S}(x|1)\beta}{(\epsilon*\alpha)P_{X|S}(x|1)(1-\beta) + P_{X|S}(x|1)\beta} \\ &= \frac{\beta}{(\epsilon*\alpha)(1-\beta)+\beta},\end{aligned}$$

where (a) follows from the law of total probability, (b) follows from the definition of conditional probability, and (c) follows from Equations (7.2) and (7.9).

Denote the first coordinate of the random variable $\tilde{Y}$ by $\tilde{Y}_1 \equiv \tilde{X} \oplus X$, and the first coordinate of $\tilde{y}$ by y_1. The same notation is used also for $\tilde{S}$ and $\tilde{s}$. Since $\frac{P_{X,S}(x,1)}{P_{X,Y}(x,1)}$ is not a function of x and is in $[0,1]$, we can define W as following:

$$W(\tilde{s}|\tilde{y}) \triangleq \begin{cases} 1 & \text{if } \tilde{s}_1 = \tilde{y}_1 \text{ and } (s,y) = (0,0) \\ 1 - \frac{P_{X,S}(x,1)}{P_{X,Y}(x,1)} & \text{if } \tilde{s}_1 = \tilde{y}_1 \text{ and } (s,y) = (0,1) \\ \frac{P_{X,S}(x,1)}{P_{X,Y}(x,1)} & \text{if } \tilde{s}_1 = \tilde{y}_1 \text{ and } (s,y) = (1,1) \\ 0 & \text{otherwise.} \end{cases}$$

We show next that Eq. (7.8) holds for W defined above:

$$\begin{aligned}
&\sum_{\tilde{y}\in\{0,1\}^2} P_{\tilde{Y}|\tilde{X}}(\tilde{y}|\tilde{x})W(\tilde{s}|\tilde{y}) = \sum_{\tilde{y}\in\{0,1\}^2} P_{\tilde{Y}_1,Y|\tilde{X}}(\tilde{y}_1, y|\tilde{x})W(\tilde{s}|\tilde{y}) \\
&\overset{\text{(d)}}{=} \sum_{\tilde{y}\in\{0,1\}^2} P_{X,Y|\tilde{X}}(\tilde{y}_1 \oplus \tilde{x}, y|\tilde{x})W(\tilde{s}|\tilde{y}) \\
&\overset{\text{(e)}}{=} \sum_{\tilde{y}\in\{0,1\}^2} P_{X,Y}(\tilde{y}_1 \oplus \tilde{x}, y)W(\tilde{s}|\tilde{y}) \\
&= \big[P_{X,Y}(\tilde{s}_1 \oplus \tilde{x}, 0)W(\tilde{s}_1, 0|\tilde{s}_1, 0) + P_{X,Y}(\tilde{s}_1 \oplus \tilde{x}, 1)W(\tilde{s}_1, 0|\tilde{s}_1, 1)\big]\mathbb{1}(s=0) \\
&\quad + P_{X,Y}(\tilde{s}_1 \oplus \tilde{x}, 1)W(\tilde{s}_1, 1|\tilde{s}_1, 1)\mathbb{1}(s=1) \\
&= \left[P_{X,Y}(\tilde{s}_1 \oplus \tilde{x}, 0) + P_{X,Y}(\tilde{s}_1 \oplus \tilde{x}, 1)\left(1 - \frac{P_{X,S}(\tilde{s}_1 \oplus \tilde{x}, 1)}{P_{X,Y}(\tilde{s}_1 \oplus \tilde{x}, 1)}\right)\right] \cdot \mathbb{1}(s=0) \\
&\quad + P_{X,Y}(\tilde{s}_1 \oplus \tilde{x}, 1)\frac{P_{X,S}(\tilde{s}_1 \oplus \tilde{x}, 1)}{P_{X,Y}(\tilde{s}_1 \oplus \tilde{x}, 1)}\mathbb{1}(s=1) \\
&= [P_{X,Y}(\tilde{s}_1 \oplus \tilde{x}, 0) + P_{X,Y}(\tilde{s}_1 \oplus \tilde{x}, 1) \\
&\quad - P_{X,S}(\tilde{s}_1 \oplus \tilde{x}, 1)]\mathbb{1}(s=0) + P_{X,S}(\tilde{s}_1 \oplus \tilde{x}, 1)\mathbb{1}(s=1) \\
&\overset{\text{(f)}}{=} [P_X(\tilde{s}_1 \oplus \tilde{x}) - P_{X,S}(\tilde{s}_1 \oplus \tilde{x}, 1)]\mathbb{1}(s=0) + P_{X,S}(\tilde{s}_1 \oplus \tilde{x}, 1)\mathbb{1}(s=1) \\
&\overset{\text{(g)}}{=} P_{X,S}(\tilde{s}_1 \oplus \tilde{x}, 0)\mathbb{1}(s=0) + P_{X,S}(\tilde{s}_1 \oplus \tilde{x}, 1)\mathbb{1}(s=1) = P_{X,S}(\tilde{s}_1 \oplus \tilde{x}, s) \\
&\overset{\text{(h)}}{=} P_{X,S|\tilde{X}}(\tilde{s}_1 \oplus \tilde{x}, s|\tilde{x}) \\
&= P_{\tilde{S}_1,S|\tilde{X}}(\tilde{s}_1, s|\tilde{x}) = P_{\tilde{S}|\tilde{X}}(\tilde{s}|\tilde{x}),
\end{aligned}$$

where (d) follows from the fact that $\tilde{Y}_1 = \tilde{X} \oplus X$, (e) follows from the independence of (X, Y) and $\tilde{X}$, (f) and (g) follow from the law of total probability, and (h) follows from the independence of (X, S) and $\tilde{X}$. So the channel W satisfies Equation (7.8) and thus the lemma holds. $\square$

We note that besides Lemma 32, the rest of the proof of Theorem 29 holds for any binary-input Gelfand-Pinsker problem. Therefore, to apply Construction 6 to other binary-input Gelfand-Pinsker problems, only an appropriate analogue of Lemma 32 needs to be proven. This also extends to Wyner-Ziv problems which are dual to the Gelfand-Pinsker problems.

7.4.2 The Rate of the Main Block

We need to show that $\lim_{n\to\infty} k/n = (1-\beta)[H(\alpha * \epsilon) - H(\alpha)]$. By [28, Equations (38) and (39)], we have $\lim_{n\to\infty} |B|/n = H(X|S)$, and $\lim_{n\to\infty} |A^c|/n = H(X|Y)$. So we get

$$\begin{aligned}\lim_{n\to\infty} k/n = \lim_{n\to\infty} |A \cap B|/n \geqslant \lim_{n\to\infty} (|B| - |A^c|)/n \\ = H(X|S) - H(X|Y).\end{aligned}$$

Heegard showed indirectly in [27] that $H(X|S) - H(X|Y) = (1-\beta)[H(\alpha * \epsilon) - H(\alpha)]$. This is enough to complete the proof of claim 1 in Theorem 29. However, we see value in seeing this relation directly, and therefore we provide a direct proof of this relation below.

Proof: We need to show that $H(X|S) - H(X|Y) = (1-\beta)[H(\alpha * \epsilon) - H(\alpha)]$. Given the distributions P_S and $P_{X|S}$, the conditional entropy $H(X|S)$ is

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

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

$$
\begin{aligned}
P_Y(0) = &\sum_{x\in\{0,1\}} P_{Y|XS}(0|x,0)P_{X|S}(x|0)P_S(0) \\
=&(1-\beta)((1-\alpha)(1-\epsilon) + \alpha\epsilon) \\
=&(1-\beta)(\alpha * (1-\epsilon)), \\
P_Y(1) =&1 - P_Y(0) \\
=&(1-\beta)(\alpha * \epsilon) + \beta.
\end{aligned}
$$

The conditional distribution $P_{X|Y}$ is given by

$$\begin{aligned}
P_{X|Y}(1|0) &= \sum_{s\in\{0,1\}} P_{XS|Y}(1,s|0) \\
&= \sum_{s\in\{0,1\}} \frac{P_{Y|XS}(0|1,s)P_{XS}(1,s)}{P_Y(0)} \\
&= \sum_{s\in\{0,1\}} \frac{P_{Y|XS}(0|1,s)P_{X|S}(1|s)P_S(s)}{P_Y(0)} \\
&= \frac{\alpha\epsilon}{\alpha * (1-\epsilon)}, \\
P_{X|Y}(1|1) &= \sum_{s\in\{0,1\}} P_{XS|Y}(1,s|1) \\
&= \sum_{s\in\{0,1\}} \frac{P_{Y|XS}(1|1,s)P_{XS}(1,s)}{P_Y(1)} \\
&= \sum_{s\in\{0,1\}} \frac{P_{Y|XS}(1|1,s)P_{X|S}(1|s)P_S(s)}{P_Y(1)} \\
&= \frac{(1-\alpha)\epsilon(1-\beta) + \frac{\epsilon(1-\alpha)}{\epsilon*\alpha}\beta}{(1-\beta)(\alpha*\epsilon)+\beta} \\
&= \frac{\epsilon(1-\alpha)}{\epsilon * \alpha}.
\end{aligned}$$

Therefore we have

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

and then

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

$\square$

7.5 Probability of Decoding Error

In this section we show that the probability of decoding error is $2^{-O(n^{1/2-\delta'})}$ for any $\delta' > \delta > 0$, and therefore this probability can be made arbitrarily small. This will complete the proof of Theorem 29.

Let $\mathcal{E}_i$ be the set of pairs of vectors (u_1^n, y_1^n) such that $\hat{u}_1^n$ is a result of decoding y_1^n and we have $\hat{u}_1^{i-1} = u_1^{i-1}$ and $\hat{u}_i \neq u_i$. The block decoding error event is given by $\mathcal{E} \equiv \cup_{i \in A \cap B} \mathcal{E}_i$. Under decoding given in (2) with an arbitrary tie-breaking rule, every $(u_1^n, y_1^n) \in \mathcal{E}_i$ satisfies

$$
P_{U_i|U_1^{i-1},Y_1^n}(u_i|u_1^{i-1}, y_1^n) \leqslant P_{U_i|U_1^{i-1},Y_1^n}(u_i \oplus 1|u_1^{i-1}, y_1^n). \tag{7.10}
$$

Consider the block decoding error probability $P_e(\lambda_{\mathcal{I}_S^c} \setminus (A \cap B)$ for a set $\lambda_{\mathcal{I}_S^c \setminus (A\cap B)}$.

For a state sequence s_1^n and the encoding rule (1), each vector u_1^n appears with probability

$$2^{-k}\left(\prod_{i\in\mathcal{I}_S} P_{U_i|U_1^i-1,S_1^n}(u_i|u_1^{i-1},s_1^n)\right)\cdot\mathbb{1}\left[\cap_{i\in\mathcal{I}_S^c\setminus(A\cap B)}\left\{\lambda_i(u_1^{i-1})=u_i\right\}\right].$$

By the definition of conditional probability and the law of total probability, the probability of error $P_e(\lambda_{\mathcal{I}_S^c\setminus(A\cap B)})$ is given by

$$\begin{aligned}P_e(\lambda_{\mathcal{I}_S^c\setminus(A\cap B)}) = \sum_{u_1^n,s_1^n,y_1^n} & 2^{-k}\left(\prod_{i\in\mathcal{I}_S} P_{U_i|U_1^i-1,S_1^n}(u_i|u_1^{i-1},s_1^n)\right)\\ & \cdot\mathbb{1}\left[\cap_{i\in\mathcal{I}_S^c\setminus(A\cap B)}\left\{\lambda_i(u_1^{i-1})=u_i\right\}\right]\\ & P_{S_1^n}(s_1^n)\cdot P_{Y_1^n|U_1^n,S_1^n}(y_1^n|u_1^n,s_1^n)\mathbb{1}[(u_1^n,y_1^n)\in\mathcal{E}].\end{aligned}$$

Now assume that the Boolean functions are drawn from the distribution

$$P\left(\cap_{i\in\mathcal{I}_S^c\setminus(A\cap B)}\left\{\lambda_i(u_1^{i-1})=u_i\right\}\right)=\prod_{i\in\mathcal{I}_S^c\setminus(A\cap B)} P_{U_i|U_1^{i-1}}(1|u_1^{i-1}). \tag{7.11}$$

We will now calculate the expectation of the decoding error probability over the random set Boolean function. We will show that this expectation is small, and this will imply that there exist at least one set of functions with small probability of decoding error. The expectation is obtained as

$$\begin{aligned}E_{\lambda_{\mathcal{I}_S^c\setminus(A\cap B)}}[P_e(\lambda_{\mathcal{I}_S^c\setminus(A\cap B)})] = \sum_{u_1^n,s_1^n,y_1^n} & \mathbb{1}[(u_1^n,y_1^n)\in\mathcal{E}]P_{S_1^n}(s_1^n)2^{-k}\\ & \left(\prod_{i\in\mathcal{I}_S} P_{U_i|U_1^{i-1},S_1^n}(u_i|u_1^{i-1},s_1^n)\right)\\ & \cdot\left(\prod_{i\in\mathcal{I}_S^c\setminus(A\cap B)} P_{U_i|U_1^{i-1}}(u_i|u_1^{i-1})\right)\cdot P_{Y_1^n|U_1^n,S_1^n}(y_1^n|u_1^n,s_1^n).\end{aligned}$$

Define the joint distribution

$$Q_{S_1^n,U_1^n,Y_1^n} \equiv 2^{-k}\left(\prod_{i\in\mathcal{I}_S} P_{U_i|U_1^{i-1},S_1^n}(u_i|u_1^{i-1},s_1^n)\right)\cdot\left(\prod_{i\in\mathcal{I}_S^c\setminus(A\cap B)} P_{U_i|U_1^{i-1}}(u_i|u_1^{i-1})\right) P_{S_1^n}(s_1^n)$$
$$\cdot P_{Y_1^n|U_1^n,S_1^n}(y_1^n|u_1^n,s_1^n).$$

Then we have

$$\begin{aligned} E_{\Lambda_{\mathcal{I}_S^c\setminus(A\cap B}}[P_e(\Lambda_{\mathcal{I}_S^c\setminus(A\cap B)})] = Q_{U_1^n,Y_1^n}(\mathcal{E}) &\leqslant \|Q_{U_1^n,Y_1^n} - P_{U_1^n,Y_1^n}\| + P_{U_1^n,Y_1^n}(\mathcal{E}) \\ &\leqslant \|Q_{U_1^n,Y_1^n} - P_{U_1^n,Y_1^n}\| + \sum_{i\in A} P_{U_1^n,Y_1^n}(\mathcal{E}_i), \end{aligned}$$

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

$$\begin{aligned} P_{U_1^n,Y_1^n}(\mathcal{E}_i) &\leqslant \sum_{u_1^i,y_1^n} P(u_1^{i-1},y_1^n)P(u_i|u_1^{i-1},y_1^n)\mathbb{1}[P(u_i|u_1^{i-1},y_1^n) \leqslant P(u_i\oplus 1|u_1^{i-1},y_1^n)] \\ &\leqslant \sum_{u_1^i,y_1^n} P(u_1^{i-1},y_1^n)P(u_i|u_1^{i-1},y_1^n)\sqrt{\frac{P(u_i\oplus 1|u_1^{i-1},y_1^n)}{P(u_i|u_1^{i-1},y_1^n)}} \\ &= Z(U_i|U_1^{i-1},Y_1^n) \\ &\leqslant 2^{-n^{1/2-\delta}}, \end{aligned}$$

where the last inequality follows from the fact that i is in the set A and therefore also in the set A.

To prove that $E_{\lambda_{\mathcal{I}_S^c\setminus(A\cap B}}[P_e(\lambda_{\mathcal{I}_S^c\setminus(A\cap B)})] \leqslant 2^{-n^{1/2-\delta}}$, we are left with showing that

$\|P_{U_1^n,Y_1^n} - Q_{U_1^n,Y_1^n}\| \leqslant 2^{-n^{1/2-\delta}}$. Notice that

$$\begin{aligned}
2\|P_{U_1^n,Y_1^n} - Q_{U_1^n,Y_1^n}\| &= \sum_{u_1^n,y_1^n} |P(u_1^n, y_1^n) - Q(u_1^n, y_1^n)| \\
&= \sum_{u_1^n,y_1^n} \left| \sum_{s_1^n} [P(s_1^n, u_1^n, y_1^n) - Q(s_1^n, u_1^n, y_1^n)] \right| \\
&\leqslant \sum_{s_1^n,u_1^n,y_1^n} |P(s_1^n, u_1^n, y_1^n) - Q(s_1^n, u_1^n, y_1^n)| \\
&= 2\|P_{S_1^n,U_1^n,Y_1^n} - Q_{S_1^n,U_1^n,Y_1^n}\|,
\end{aligned}$$

where the inequality follows from the triangular inequality. The following lemma completes the proof of Claim 2.

Lemma 33.

$$\|P_{S_1^n,U_1^n,Y_1^n} - Q_{S_1^n,U_1^n,Y_1^n}\| \leqslant 2^{-n^{1/2-\delta}}. \tag{7.12}$$

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

$$
\begin{aligned}
&2\|P_{S_1^n,U_1^n,Y_1^n} - Q_{S_1^n,U_1^n,Y_1^n}\| \\
&\quad = \sum_{s_1^n,u_1^n,y_1^n} |P(s_1^n,u_1^n,y_1^n) - Q(s_1^n,u_1^n,y_1^n)| \\
&\quad \overset{\text{(a)}}{=} \sum_{s_1^n,u_1^n,y_1^n} |P(u_1^n|s_1^n) - Q(u_1^n|s_1^n)| P(s_1^n) P(y_1^n|u_1^n,s_1^n) \\
&\quad \overset{\text{(b)}}{=} \sum_{s_1^n,u_1^n,y_1^n} \left|\prod_{i=1}^n P(u_i|u_1^{i-1},s_1^n) - \prod_{i=1}^n Q(u_i|u_1^{i-1},s_1^n)\right| P(s_1^n)P(y_1^n|u_1^n,s_1^n) \\
&\quad \overset{\text{(c)}}{=} \sum_{s_1^n,u_1^n,y_1^n} \left|\sum_i [P(u_i|u_1^{i-1},s_1^n) - Q(u_i|u_1^{i-1},s_1^n)]\right| P(s_1^n) \\
&\qquad \cdot \prod_{j=1}^{i-1} P(u_j|u_1^{j-1},s_1^n) \prod_{j=i+1}^{n} Q(u_j|u_1^{j-1},s_1^n) P(y_1^n|u_1^n,s_1^n) \\
&\quad \overset{\text{(d)}}{\leqslant} \sum_{i\in\mathcal{I}_S^c} \sum_{s_1^n,u_1^n,y_1^n} \left|P(u_i|u_1^{i-1},s_1^n) - Q(u_i|u_1^{i-1},s_1^n)\right| P(s_1^n) \\
&\qquad \cdot \prod_{j=1}^{i-1} P(u_j|u_1^{j-1},s_1^n) \prod_{j=i+1}^{n} Q(u_j|u_1^{j-1},s_1^n) P(y_1^n|u_1^n,s_1^n) \\
&\quad = \sum_{i\in\mathcal{I}_S^c} \sum_{s_1^n,u_1^i} \left|[P(u_i|u_1^{i-1},s_1^n) - Q(u_i|u_1^{i-1},s_1^n)]\right| \prod_{j=1}^{i-1} P(u_j|u_1^{j-1},s_1^n) P(s_1^n) \\
&\quad \overset{\text{(e)}}{=} \sum_{i\in\mathcal{I}_S^c} \sum_{s_1^n,u_1^{i-1}} P(u_1^{i-1},s_1^n) 2\|P_{U_i|U_1^{i-1}=u_1^{i-1},S_1^{i-1}=s_1^{i-1}} - Q_{U_i|U_1^{i-1}=u_1^{i-1},S_1^{i-1}=s_1^{i-1}}\| \\
&\quad \overset{\text{(f)}}{\leqslant} \sum_{i\in\mathcal{I}_S^c} \sum_{s_1^n,u_1^{i-1}} P(u_1^{i-1},s_1^n)\sqrt{2\ln 2}\sqrt{D(P_{U_i|U_1^{i-1}=u_1^{i-1},S_1^{i-1}=s_1^{i-1}}\|Q_{U_i|U_1^{i-1}=u_1^{i-1},S_1^{i-1}=s_1^{i-1}})} \\
&\quad \overset{\text{(g)}}{\leqslant} \sum_{i\in\mathcal{I}_S^c} \sqrt{(2\ln 2)\sum_{s_1^n,u_1^{i-1}} P(u_1^{i-1},s_1^n) D(P_{U_i|U_1^{i-1}=u_1^{i-1},S_1^{i-1}=s_1^{i-1}}\|Q_{U_i|U_1^{i-1}=u_1^{i-1},S_1^{i-1}=s_1^{i-1}})} \\
&\quad = \sum_{i\in\mathcal{I}_S^c} \sqrt{(2\ln 2) D(P_{U_i}\|Q_{U_i}|U_1^{i-1},S_1^{i-1})} \\
&\quad \overset{\text{(h)}}{=} \sum_{i\in A\cap B} \sqrt{(2\ln 2)[1 - H(U_i|U_1^{i-1},S_1^n)]} \\
&\qquad + \sum_{i\in\mathcal{I}_S^c\setminus(A\cap B)} \sqrt{(2\ln 2)[H(U_i|U_1^{i-1}) - H(U_i|U_1^{i-1},S_1^n)]}, \qquad (7.13)
\end{aligned}
$$

where

(a) follows from the fact that $P(s_1^n) = Q(s_1^n)$ and $P(y_1^n|u_1^n, s_1^n) = Q(y_1^n|u_1^n, s_1^n)$,

(b) follows from the chain rule,

(c) follows from the telescoping expansion

$$\begin{aligned} B_1^n - A_1^n &= \sum_{i=1}^n A_1^{i-1} B_i^n - \sum_{i=1}^n A_1^i B_{i+1}^n \\ &= \sum_{i=1}^n (B_i - A_i) A_1^{i-1} B_{i+1}^n, \end{aligned}$$

where A_j^k and B_j^k denote the products $\prod_{i=j}^k A_i$ and $\prod_{i=j}^k B_i$, respectively,

(d) follows from the triangular inequality and the fact that $P(u_i|u_1^i - 1, s_1^n) = Q(u_i|u_1^i - 1, s_1^n)$ for all $i \in \mathcal{I}_S$,

(e) follows from the chain rule again,

(f) follows from Pinsker's inequality (see, e.g., [15, Lemma 11.6.1]),

(g) follows from Jensen's inequality and

(h) follows from the facts that $Q(u_i|u_1^{i-1}) = 1/2$ for $i \in A \cap B$ and $Q(u_i|u_1^{i-1}) = P(u_i|u_1^{i-1})$ for $i \in \mathcal{I}_S^c \setminus (A \cap B)$.

Now if $i \in A \cap B$, we have

$$\begin{aligned} 1 - H(U_i|U_1^{i-1}, S_1^n) &\leqslant 1 - [Z(U_i|U_1^{i-1}, S_1^n)]^2 \\ &\leqslant 2^{-2n^{1/2-\delta}}, \end{aligned} \tag{7.14}$$

where the first inequality follows from Proposition 28 and the second inequality follows from the fact the i is in B. In addition, if $i \in \cap\mathcal{I}_S^c \setminus (A \cap B)$, it follows that

$$\begin{aligned} H(U_i|U_1^{i-1}) - H(U_i|U_1^{i-1}, S_1^n) \leqslant & Z(U_i|U_1^{i-1}) - (Z(U_i|U_1^{i-1}, S_1^n))^2 \\ \leqslant & \min\left\{Z(U_i|U_1^{i-1}), 1 - (Z(U_i|U_1^{i-1}, S_1^n))^2\right\} \\ \leqslant & 2 \cdot 2^{-n^{1/2-\delta}}, \end{aligned} \tag{7.15}$$

where the last inequality follows from the fact that i is in $\mathcal{I}_S^c$. This completes the proof of the lemma. □

The proof of claim 3 follows by using Lemma 33 and an argument similar to the proofs of Theorem 1 and Lemma 2 in [10].

7.6 Discussion

We presented a capacity-achieving coding scheme for the write-once memory with errors in non-stuck cells. The coding scheme can used directly for a memory in which the stuck cells are caused by manufacturing faults of memory wear. For a theoretical model of WOM, the i.i.d. assumption on the stuck cell does not hold, since the memory state is determined by previous writing rounds. This issue can be solved by sharing randomness between the encoder and decoder, as in modifications M3 and M4 in [10, Section IV].

8. POLAR CHANNEL CODES IN MULTI-LEVEL FLASH MEMORIES

The urgency for improving the reliability of flash memories calls for continuously searching for optimal signal detection algorithms as well as optimal channel coding schemes. This chapter focuses on the latter, and studies the practical approaches to polar coding for multi-level flash memories.

Polar codes were first proposed by Arıkan [2] is the first class of capacity-achieving codes with explicit construction. Their universality [26], effective code construction [55] [69] and decoding algorithms [29] [68], as well as implementations [46] [56] make them a potential candidate for optimal channel coding schemes. However, the practical performance of polar codes in flash memories is still unknown, and applying polar codes to flash channels presents many important challenges. For instance, polar codes require the code length to be an integer power of two which does not fit in flash pages of different sizes. To conduct rigorous experimental analysis, the decoding performance of polar codes need to be compared with that of other ECCs on the same random input and output data sets from flash characterization platforms. Such testing data are not assumed to be the codewords of any ECC. Moreover, the construction of polar codes uses the channel statistics, and one concern is that new polar codes need to be frequently constructed for optimized performance when flash memory endures and the channel gradually degrades, which is prohibitively expensive in practice. Motivated by these challenges, we report part of the efforts towards realizing polar channel decoders for flash memories.

To make polar codewords fit different page sizes of flash memories, length-adapted codes are needed. Punctured polar codes have been studied recently [20] [65]. Puncturing removes the selected codeword symbols before transmission. The location of the removed symbols are known both to the encoder and the decoder. After the codeword is received, the punctured locations are filled with erasures. Puncturing has low implementation complexity, but degrades decoding performance due to the additional erasures introduced. This work explores an alternative approach for length-adapted polar codes through shortening. Shortening has been widely used for existing codes such as BCH codes and low density parity check (LDPC) codes in flash memories. We propose the schemes for shortening both non-systematic [2] and systematic polar codes [4]. Shortening obtains a shorter codeword by assigning selected codeword symbols of the longer codeword to predetermined values made known both to the encoder and the decoder. The selected symbols are removed before transmission and inserted back before decoding. Since the symbols inserted are always correct, shortening does not introduce additional noise.

Rate-compatible polar codes can be implemented by adjusting the size of frozen sets without constructions of new codes [20] thanks to the property that reliability orders of the subchannels of polar codes is preserved for degrading and upgrading channels [45] [52]. We show that this property guarantees the feasibility of a practical adaptive polar decoding framework for flash channels. The decoder adaptively switches to use lower code rates as flash memory endures, and the code of each rate is used for a continuous range of channel parameters (it is not affordable in practice to immediately switch the rate of a code once channel condition slightly changes). We prove that repeatedly polar code construction is not necessary for such adaptive decoders. With the codes constructed for practical flash channels, we observed the order preservation of subchannel reliability, and our extensive experiments further demonstrate that the decoding performance by using one code closely approaches the optimized performance by constructing codes for different channel parameters.

8.1 The Models of Flash Memories

This section discusses the models of flash memories. We focus on the channels consisted of MLCs, and model the noise for such channels.

8.1.1 Reading MLC

Data are read either through hard or soft sensing. In each approach, MSB and LSB are read independently. Hard sensing returns (possibly noisy) the value of the bit being read, while soft sensing outputs the log-likelihood ratio (LLR) of the bit. Specifically, hard sensing uses one reference threshold voltage between two adjacent distributions as shown in Figure 1.3. Let the reference threshold voltages be $V_{\mathrm{th},1}$, $V_{\mathrm{th},2}$, and $V_{\mathrm{th},3}$. To read LSB, the cell threshold voltage V_{th} is compared with $V_{\mathrm{th},2}$, returning 0 if $V_{\mathrm{th}} > V_{\mathrm{th},2}$, and 1 otherwise. To read MSB utilizes both $V_{\mathrm{th},1}$ and $V_{\mathrm{th},3}$: when $V_{\mathrm{th},1} < V_{\mathrm{th}} < V_{\mathrm{th},3}$, output 0, otherwise output 1. Soft sensing use k reference threshold voltages between two adjacent distributions [73]. Figure 8.1 shows an example for $k = 3$. For $i \in \{1, 2, 3\}$,

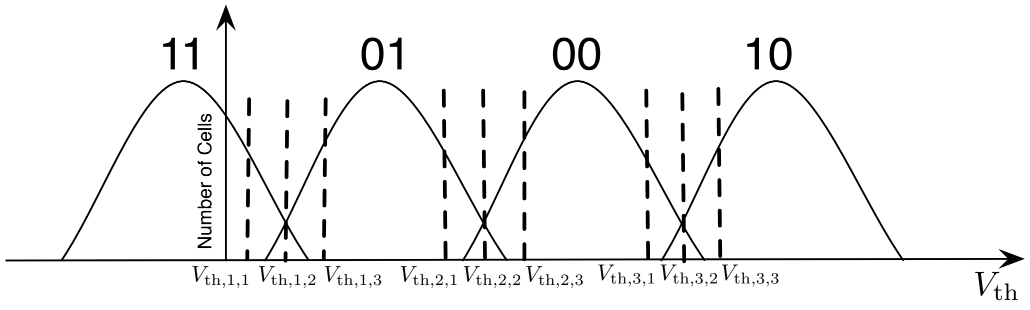


Figure 8.1: An example of soft sensing with $k = 3$.

$j \in \{1, 2, \cdots, k\}$, let $V_{\mathrm{th},i,j}$ be the j-th reference threshold voltage of the k reference threshold voltages between level i and $i + 1$. The threshold voltage range is thus divided into $3k + 1$ bins. During reading, the bin that V_{th} falls into is determined after sensing with the reference threshold voltages, according to which the LLRs are computed. Assume V_{th} belongs to the bin of the interval $[V_{\mathrm{th},i}, V_{\mathrm{th},i+1}]$, the LLRs of the LSB and the MSB are given below

$$\begin{aligned} L_{\mathrm{lsb}} &\triangleq \ln \frac{\mathrm{P}(V \in [V_{\mathrm{th},i}, V_{\mathrm{th},i+1}) \mid \mathrm{LSB}=1)}{\mathrm{P}(V \in [V_{\mathrm{th},i}, V_{\mathrm{th},i+1}) \mid \mathrm{LSB}=0)} \\ &= \ln \frac{\sum_{l\in\{3,4\}} \mathrm{P}(V \in [V_{\mathrm{th},i}, V_{\mathrm{th},i+1}) \mid l)}{\sum_{l\in\{1,2\}} \mathrm{P}(V \in [V_{\mathrm{th},i}, V_{\mathrm{th},i+1}) \mid l)} \\ &= \ln \frac{Q(\frac{V_{\mathrm{th},i}-\mu_3}{\sigma_3}) - Q(\frac{V_{\mathrm{th},i+1}-\mu_3}{\sigma_3}) + Q(\frac{V_{\mathrm{th},i}-\mu_4}{\sigma_4}) - Q(\frac{V_{\mathrm{th},i+1}-\mu_4}{\sigma_4})}{Q(\frac{V_{\mathrm{th},i}-\mu_1}{\sigma_1}) - Q(\frac{V_{\mathrm{th},i+1}-\mu_1}{\sigma_1}) + Q(\frac{V_{\mathrm{th},i}-\mu_2}{\sigma_2}) - Q(\frac{V_{\mathrm{th},i+1}-\mu_2}{\sigma_2})} \end{aligned}$$

$$\begin{aligned} L_{\mathrm{msb}} &\triangleq \ln \frac{\mathrm{P}(V \in [V_{\mathrm{th},i}, V_{\mathrm{th},i+1}) \mid \mathrm{MSB}=1)}{\mathrm{P}(V \in [V_{\mathrm{th},i}, V_{\mathrm{th},i+1}) \mid \mathrm{MSB}=0)} \\ &= \ln \frac{\sum_{l\in\{1,4\}} \mathrm{P}(V \in [V_{\mathrm{th},i}, V_{\mathrm{th},i+1}) \mid l)}{\sum_{l\in\{2,3\}} \mathrm{P}(V \in [V_{\mathrm{th},i}, V_{\mathrm{th},i+1}) \mid l)} \\ &= \ln \frac{Q(\frac{V_{\mathrm{th},i}-\mu_1}{\sigma_1}) - Q(\frac{V_{\mathrm{th},i+1}-\mu_1}{\sigma_1}) + Q(\frac{V_{\mathrm{th},i}-\mu_4}{\sigma_4}) - Q(\frac{V_{\mathrm{th},i+1}-\mu_4}{\sigma_4})}{Q(\frac{V_{\mathrm{th},i}-\mu_2}{\sigma_2}) - Q(\frac{V_{\mathrm{th},i+1}-\mu_2}{\sigma_2}) + Q(\frac{V_{\mathrm{th},i}-\mu_3}{\sigma_3}) - Q(\frac{V_{\mathrm{th},i+1}-\mu_3}{\sigma_3})} \end{aligned}$$

where the function $Q(\cdot)$ is the Q-function of a Gaussian distribution. The sign of the LLR represents represents determines the value of the bit that is more likely to be, and the magnitude measures the level of confidence.

8.1.2 A Cascaded Noise Channel Model

We model the noise introduced during writing and reading MLCs with a cascaded noise channel. A cascaded channel consists of more than one subchannels where two adjacent subchannels are connected such that the output of the first subchannel are the input of the second subchannel. Cascaded noise channel is studied for LDPC codes where the channel is motivated by the errors introduced during writing and reading for magnetic recording channels [33]. In this work, the cascaded noise channel for each bit in a MLC has two binary symmetric channels (BSCs) with cross-over probabilities p_{r} and p_{w} shown in Figure 8.2. We refer the first BSC as write channel and the second BSC as read channel which we describe below.

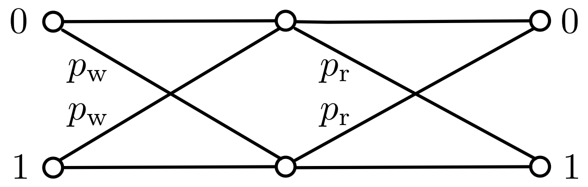


Figure 8.2: The cascaded noise channel of MLCs.

The write channel is motivated by the errors introduced during programming such as misprogram errors and stuck cells. During two step programming, the value of LSB is read from cell when MSB is programmed. The LSB can be misread when the cell threshold voltage after first step falls into the unreliable region where the distribution of the erased state and the intermediate state overlaps. Given the incorrect value of LSB, and the value of the MSB to be programmed, the cell will be misprogrammed to a undesired state. Stuck cells mostly happen when the program-erase cycles approach the endurance of the cells. A stuck cell stay at some level and can not be changed during the programming. Therefore, when the MSB or the LSB to be written to a stuck cell does not equal to the bits represented by the stuck level, error will occur. Since the error patterns in write channel for MSB and LSB include $0 \to 1$ and $1 \to 0$, it is a reasonable approximation to use BSCs as the first-order approximation to the write channel.

The read channel is implied by the cell level distributions of MLCs. When the threshold voltage of a cell is at the region where two distributions overlaps, the cell will be misread with high probability. The cross-over probabilities of the read channel are computed according to different sensing methods using the Q functions of the threshold distributions and the reference threshold voltages. Due to space limit, we skip the computations here.

In practice, the standard deviation of each threshold voltage distribution is close to each other. For the rest of the chapter, we assume all the threshold voltage distributions share the same standard deviation σ, and the read channel is a binary symmetric channel.

Corollary 34. *Let the frozen sets of the polar codes achieving the capacities of the write channel W_{w}, the read channel W_{r} and the cascaded channel W_{c} be F_{w}, F_{r} and F_{c}, respectively. Then $F_{\mathrm{w}} \subseteq F_{\mathrm{c}}$ and $F_{\mathrm{r}} \subseteq F_{\mathrm{c}}$.*

Proof: The transition probability of the cascaded channel $p_{\mathrm{c}} \triangleq (1-p_w)p_r + p_w(1-p_r) = p_r + p_w - 2p_w p_r$. Since $p_w, p_r \leqslant \frac{1}{2}$, $p_{\mathrm{w}} \leqslant p_{\mathrm{c}}$ and $p_{\mathrm{r}} \leqslant p_{\mathrm{c}}$, we have $W_{\mathrm{c}} \succeq W_{\mathrm{w}}$, and $W_{\mathrm{c}} \succeq W_{\mathrm{r}}$. According to Lemma **??**, $F_{\mathrm{w}} \subseteq F_{\mathrm{c}}$ and $F_{\mathrm{r}} \subseteq F_{\mathrm{c}}$. □

The corollary states that the polar code constructed using the channel statistics of the cascaded channel corrects the errors introduced by the write channel and the read channel.

8.2 Polar Codes for Flash Memories

This section proposes the schemes for shortening non-systematic and systematic polar codes for flash memories with different page sizes. The performance of shortened polar codes is further evaluated using random input and output data obtained from a flash characterization platform.

8.2.1 Shortened Non-systematic Polar Codes

Polar codes require the code length be 2^m where m is an integer. Without length-adaptation a codeword does not directly fit in flash memories of typical page sizes. For instance, a common page size for multi-level flash memories is 71488 bits. A page is typically split into 4 ECC codewords with length 17872 bits, or 8 codewords with length 8936 bits. None of these lengths is an integer power of two. We study the approaches to shortened polar codes defined below:

Definition 35. *An (N, K, K')-shortened polar code (SPC) is a polar code of length $N-K'$*

obtained from an (N, K)-polar code with block length $N = 2^m$ and information bit length K by assigning K' predetermined input symbols to known values before encoding, and removing K' predetermined codeword symbols after encoding.

Let us define the notations used later in this section. Consider an (N, K) binary polar code with $N = 2^m$. Let the non-frozen set of the code be $\mathcal{A} \triangleq \{a_1, a_2, \cdots, a_K\} \subseteq \{1, 2, \cdots, N\}$, and let the frozen set $\bar{\mathcal{A}} \triangleq \{b_1, b_2, \cdots, b_{N-K}\}$ be the complement. We also assume that $a_1 < a_2 < \cdots < a_K$ and $b_1 < b_2 < \cdots < b_{N-K}$. Denote the input bits to the encoder by $\boldsymbol{u} \triangleq (u_1, u_2, \cdots, u_N) = (\boldsymbol{u}_{\mathcal{A}}, \boldsymbol{u}_{\bar{\mathcal{A}}})$ to represent $\boldsymbol{u}$, where $\boldsymbol{u}_{\mathcal{A}} \triangleq (u_i : i \in \mathcal{A})$ contains the message bits and $\boldsymbol{u}_{\bar{\mathcal{A}}} \triangleq (u_i : i \in \bar{\mathcal{A}})$ contains the frozen bits. The codeword $\boldsymbol{x} \triangleq (x_1, x_2, \cdots, x_N)$ computed by encoding is written to cells. The reading process outputs a (possibly noisy) codeword $\boldsymbol{y} \triangleq (y_1, y_2, \cdots, y_N)$, and decoder computes the estimated codeword $\hat{\boldsymbol{x}} \triangleq (\hat{x}_1, \hat{x}_2, \cdots, \hat{x}_N)$.

We first study the shortening of non-systematic polar codes (NSPCs) whose encoding of an (N, K)-NSPC follows the linear transformation $\boldsymbol{x} \triangleq \boldsymbol{u}\mathbf{G}$. Our scheme uses the following property of the generator matrix $\mathbf{G}$:

Remark 36. *The generator matrix $\mathbf{G}$ is a lower triangular matrix with ones on the diagonal.*

Lemma 37. $(u_{N-K'+1}, u_{N-K'+2}, \cdots, u_N)$ *are all* 0*s iff* $(x_{N-K'+1}, x_{N-K'+2}, \cdots, x_N)$ *are all* 0*s.*

Proof: According to Remark 36, the matrix $\mathbf{G}$ is invertible. Therefore, there is a one-to-one mapping between the $(u_{N-K'+1}, \cdots, u_N)$ and $(x_{N-K'+1}, \cdots, x_N)$, and when $(u_{N-K'+1}, \cdots, u_N)$ are all 0s, $(x_{N-K'+1}, \cdots, x_N)$ will be 0s. $\square$

The above lemma suggests we obtain an (N, K, K')-SPC from an (N, K)-NSPC by setting the last K' input bits to 0s, then removing the last K' codeword symbols after encoding. Among the K' input bits, there are K'' non-frozen bits and $K' - K''$ frozen bits where $K'' = |\{i | i \in \mathcal{A} \text{ and } N - K' + 1 \leqslant i \leqslant N\}|$.

Theorem 38. *An (N, K, K')-SPC obtained through the encoding above has rate $\frac{K-K''}{N-K'} \in [\frac{K-K'}{N-K'}, \frac{K}{N}]$.*

The encoding and the decoding algorithms are given below.

Encoding

1. For $j \in \{a_{N-K-K'+K''+1}, a_{N-K-K'+K''+2}, \cdots, a_{N-K}\}$, let $u_j = 0$. For $j \in \bar{\mathcal{A}} - \{a_{N-K-K'+K''+1}, \cdots, a_{N-K}\}$, let u_j be any predetermined frozen bit (e.g. 0), completing the frozen bits $\boldsymbol{u}_{\bar{\mathcal{A}}}$.

2. For $j \in \{b_{K-K''+1}, b_{K-K''+2}, \cdots, b_K\}$, let $u_j = 0$. Store $K - K''$ message bits in $(u_{b_1}, u_{b_2}, \cdots, u_{b_{K-K''}})$, completing the message bits $\boldsymbol{u}_{\mathcal{A}}$.

3. Compute $\boldsymbol{x} = \boldsymbol{u}\mathbf{G}$, and transmit the shortened codeword $(x_1, x_2, \cdots, x_{N-K'})$.

Decoding

1. After receiving a noisy shortened codeword $(y_1, y_2, \cdots, y_{N-K'})$, let the codeword $\boldsymbol{y} = (y_1, y_2, \cdots, y_{N-K'}, 0, \cdots, 0)$ with K' 0s in the end.

2. Correct $\boldsymbol{y}$. The decoder treats the added bits $(y_{N-K'+1}, y_{N-K'+2}, \cdots, y_N)$ as if they went through a perfect channel and have unit probability of being 0.

8.2.2 Shortened Systematic Polar Codes

In practice, flash memories prefer to use systematic codes due to its lower latency for reading information bits. Systematic polar codes (SYPCs) have been proposed recently by Arıkan [4]. In the following, we briefly review the construction, and propose the shortening scheme for SYPCs.

Definition 39. *[4] Let the sets $\mathcal{B} \subseteq \{1, 2, \cdots, N\}$, and $\bar{\mathcal{B}}$ be the complement. Therefore, $\boldsymbol{u} = (\boldsymbol{x}_{\mathcal{B}}, \boldsymbol{x}_{\bar{\mathcal{B}}})$. Let $\mathbf{G}_{\mathcal{AB}}$ be a submatrix of $\mathbf{G}$ such that for each element $G_{i,j}$, the indices $i \in \mathcal{A}$, $j \in \mathcal{B}$. For any given non-systematic polar encoder with parameter $(\mathcal{A}, \boldsymbol{u}_{\bar{\mathcal{A}}})$, a systematic polar encoder $(\mathcal{B}, \boldsymbol{u}_{\bar{\mathcal{A}}})$ exists if there is a one-to-one mapping from $\boldsymbol{u}_{\mathcal{A}}$ to $\boldsymbol{x}_{\mathcal{B}}$ following*

$$\begin{aligned} \boldsymbol{x}_{\mathcal{B}} &= \boldsymbol{u}_{\mathcal{A}} \mathbf{G}_{\mathcal{AB}} + \boldsymbol{u}_{\bar{\mathcal{A}}} \mathbf{G}_{\bar{\mathcal{A}}\mathcal{B}}, \\ \boldsymbol{x}_{\bar{\mathcal{B}}} &= \boldsymbol{u}_{\mathcal{A}} \mathbf{G}_{\mathcal{A}\bar{\mathcal{B}}} + \boldsymbol{u}_{\bar{\mathcal{A}}} \mathbf{G}_{\bar{\mathcal{A}}\bar{\mathcal{B}}}. \end{aligned} \tag{8.1}$$

Lemma 40. *[4] A systematic polar encoder defined in Eq. (8.1) exists if $\mathcal{B} = \mathcal{A}$.*

Proof: If $\mathcal{B} = \mathcal{A}$, the matrix $G_{\mathcal{AB}} = G_{\mathcal{AA}}$ is a lower-triangular invertible matrix with ones on the diagonal. Therefore, an one-to-one mapping between $\boldsymbol{u}_{\mathcal{A}}$ and $\boldsymbol{x}_{\mathcal{A}}$ always exists, and $\boldsymbol{u}_{\mathcal{A}} = (\boldsymbol{x}_{\mathcal{A}} + \boldsymbol{u}_{\bar{\mathcal{A}}} \mathbf{G}_{\bar{\mathcal{A}}\mathcal{A}}) \mathbf{G}_{\mathcal{AA}}^{-1}$. □

To shorten SYPCs, we need the following theorem:

Theorem 41. *Let $\mathcal{B} = \mathcal{A}$, and let $\boldsymbol{u}_{\bar{\mathcal{A}}}$ be all 0s. There is a one-to-one mapping between $(u_{a_{K-K'+1}}, u_{a_{K-K'+2}}, \cdots, u_{a_K})$ and $(x_{a_{K-K'+1}}, x_{a_{K-K'+2}}, \cdots, x_{a_K})$.*

Proof: The matrix $\mathbf{G}_{\mathcal{AA}}$ is a $K \times K$ lower-triangular matrix with ones on the diagonal. Let $\mathbf{G}_{\mathcal{CC}}$ be a submatrix of $\mathbf{G}_{\mathcal{AA}}$ where $\mathcal{C} = \{a_{K-K'+1}, \cdots, a_K\}$. We have

$$(x_{a_{K-K'+1}}, \cdots, x_{a_K}) = (u_{a_{K-K'+1}}, \cdots, u_{a_K}) \cdot \mathbf{G}_{\mathcal{CC}}.$$

Since $\mathbf{G}_{\mathcal{CC}}$ is a $K' \times K'$ lower-triangular matrix with ones on the diagonal, it is also invertible. This completes the proof. □

The theorem states that it is feasible to obtain an (N, K, K')-SPC from an (N, K)-SYPC by letting frozen bits be all 0s, and setting the last K' bits of $\boldsymbol{u}_{\mathcal{A}}$ to predetermined values before encoding. The last K' bits of $\boldsymbol{x}_{\mathcal{A}}$ are removed after encoding.

Theorem42. *An (N, K, K')-SYPC obtained through the encoding above has rate $\frac{K-K'}{N-K'}$.*

An instance of the encoding and the decoding algorithms for shortened SYPCs is given below, where we assign $(u_{a_{K-K'+1}}, \cdots, u_{a_K})$ to all 0s.

Encoding

1. Let $\boldsymbol{u}_{\bar{\mathcal{A}}}$ be all 0s.

2. Store $K - K'$ message bits in $(u_{a_1}, u_{a_2}, \cdots, u_{a_{K-K'}})$, and let $(u_{a_{K-K'+1}}, \cdots, u_{a_K})$ be all 0s.

3. Compute $\boldsymbol{x} = (\boldsymbol{x}_{\mathcal{A}}, \boldsymbol{x}_{\bar{\mathcal{A}}})$ using systematic encoding specified in Eq. (8.1), and transmit the shortened codeword $((x_{a_1}, x_{a_2}, \cdots, x_{a_{K-K'}}), \boldsymbol{x}_{\bar{\mathcal{A}}})$.

Decoding

1. After receiving a noisy shortened polar codeword $((y_{a_1}, y_{a_2}, \cdots, y_{a_{K-K'}}), \boldsymbol{y}_{\bar{\mathcal{A}}})$, compute $\boldsymbol{y}_{\mathcal{A}} = (y_{a_1}, y_{a_2}, \cdots, y_{a_{K-K'}}, 0, \cdots, 0)$ with K' 0s appended at the end. We obtain the unshortened codeword $\boldsymbol{y} = (\boldsymbol{y}_{\mathcal{A}}, \boldsymbol{y}_{\bar{\mathcal{A}}})$.

2. Correct $\boldsymbol{y}$ with a polar decoder with frozen bits $\boldsymbol{u}_{\bar{\mathcal{A}}}$ (all 0s), treating the bits $(y_{a_{K-K'+1}}, \cdots, y_{a_K})$ as if they went through a perfect channel and have unit probability of being 0.

8.2.3 Polar Codes with Bit-reversal Permutation

For the polar codes proposed in [2], codeword symbols are permuted by multiplying the generator matrix $\mathbf{G}$ with the bit-reversal permutation matrix $\mathbf{B}_N$. To adapt the shortening methods for the permuted codes simply requires modifying the locations of the symbols that are removed after encoding (and are inserted back before decoding): For permitted non-systematic polar codes, the K' indices of the bits that are removed are the images of the indices $(N-K'+1, N-K'+2, \cdots, N)$ under bit-reversal permutations; for permuted SYPCs, the K' indices are the images of the indices $\{a_{K-K'+1}, a_{K-K'+2}, \cdots, a_K\}$ under bit-reversal permutations.

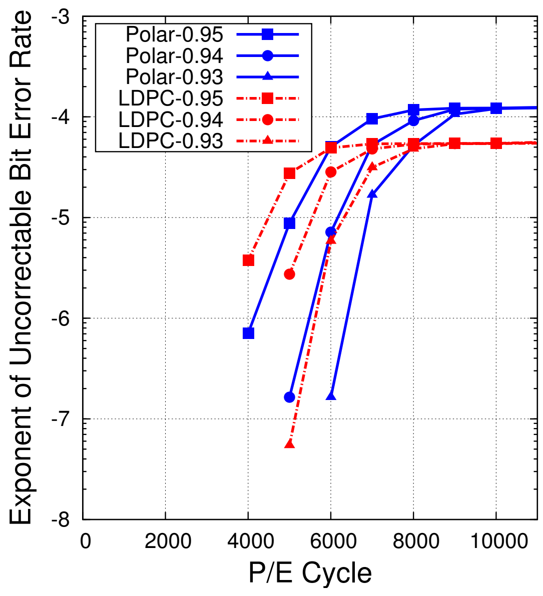
8.2.4 Performance Evaluation

We evaluated the decoding performance of shortened polar codes with the data from the characterizations of MLC flash chips using $2Y$-nm technology from some vendor. The characterization process sequentially program each page in a block with random input bits, reads the stored (and possibly noisy) data, and erase the block for the next write. Such an iteration is referred as a program/erase cycle (PEC). Raw bit error rates increase as PEC grows, and the endurance of a cell is measured by the maximum PECs the cell can carry to reliably store data. Starting with a new chip, we continue program-erase cycling the chip, recording the raw input and output data at multiple PECs during the lifetime of the block. As data written to the block are pseudo-random and are not ECC codewords, coset coding technique is needed to view such random sequences as the codewords of the ECC being evaluated. Fortunately, this is always feasible for polar codes as stated below:

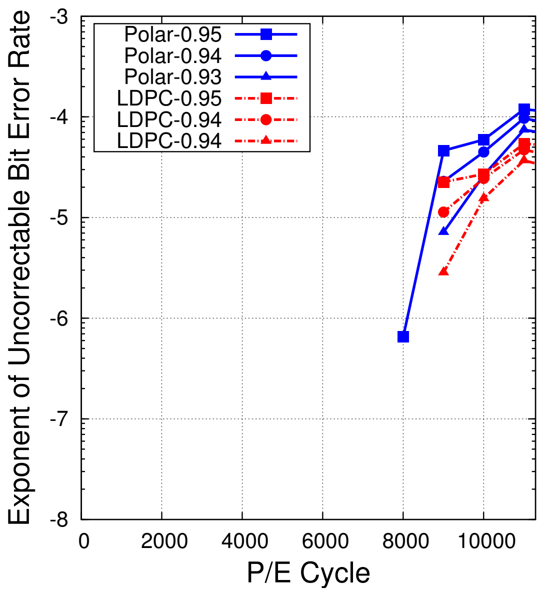
Lemma 43. *Given an (N, K)-polar code with frozen set $\bar{\mathcal{A}}$, $\forall \boldsymbol{x} \in \{0,1\}^N$, there is a unique $\boldsymbol{u}_{\mathcal{A}} \in \{0,1\}^K$ and a unique $\boldsymbol{u}_{\bar{\mathcal{A}}} \in \{0,1\}^{N-K}$ such that $\boldsymbol{x} = (\boldsymbol{u}_{\mathcal{A}}, \boldsymbol{u}_{\bar{\mathcal{A}}}) \cdot \mathbf{G}$.*

Corollary 44. *Given an (N, K, K')-SPC obtained from an (N, K)-NSPC with frozen set $\bar{\mathcal{A}}$, let $K'' = |\{i | i \in \mathcal{A}$ and $N - K' + 1 \leqslant i \leqslant N\}|$, $\forall \boldsymbol{x}' \in \{0,1\}^{N-K'}$, there is a unique $\boldsymbol{u}'_{\mathcal{A}} \in \{0,1\}^{K-K'}$ and a unique $\boldsymbol{u}'_{\bar{\mathcal{A}}} \in \{0,1\}^{N-K-K'+K''}$, such that $(\boldsymbol{x}', \underbrace{0, \cdots, 0}_{K'}) = ((\boldsymbol{u}'_{\mathcal{A}}, \underbrace{0, \cdots, 0}_{K''})_{\mathcal{A}}, (\boldsymbol{u}'_{\bar{\mathcal{A}}}, \underbrace{0, \cdots, 0}_{K'-K''})_{\bar{\mathcal{A}}}) \cdot \mathbf{G}$.*

Figure 8.3 shows the average uncorrectable bit error rates (UBERs) of shortened polar codes at different PECs with both hard sensing (Figure 8.3(a)) and soft sensing (Figure 8.3(b)). We directly use the list decoding algorithm by Tal and Vardy [68] specified in



(a) Hard Decoding



(b) Soft Decoding

Figure 8.3: The performance of polar codes and LDPC codes at different PECs.

probability domain with list size 32. The input noisy codeword bits are determined by the signs of the LLRs, and the transition probability p of the BSC that the decoder assumes a codeword bit goes through is approximated from the LLR L of the bit by:

$$p = \frac{e^{-|L|}}{1 + e^{-|L|}}.$$

We compares with the equivalent LDPC codes using min-sum decoding [50]. Three rates (0.93, 0.94 and 0.95) of interest to flash memories are used. We assume each page stores 8 length-7943 polar codewords shortened from a length-2^{13} polar code. The code is constructed using the degrading merge algorithm [69] for the BSC with the cross-over probability measured at the current PEC. The PECs when decoding failures first occur are of special interest to flash memories. The results suggest both codes have similar performance in flash memories, and soft sensing significantly improves the endurance of

MLCs (the vendor-specified endurance is 3×10^3 PECs). With hard sensing polar codes with rate 0.93 have 10^3-PEC gain comparing to LDPC codes, while LDPC codes have 10^3-PEC gain with soft sensing at rate 0.95.

We further evaluate the code performance with larger block lengths and on errors which are more symmetric. Figure 8.4a compares the soft and hard decoding performance between polar codes of code lengths 2^{13} and 2^{14}. Although the longer codes give lower UBERs, the codes at the same rates fail to decode at the same PECs. The errors produced by reading with the current realistic soft sensing are not fully symmetric (to achieve low implementation complexity), which hurts the performance of linear block codes. Figure 8.4b compares the soft decoding performance of polar codes under the current realistic soft sensing with that of using a genie soft sensing. The latter reads at all possible sensing reference threshold voltage settings and performs brute force search for the reference threshold voltages which maximize the degree of symmetry of the errors. The results show that lower BER can be achieved by making errors more symmetric. The rate-0.94 code using the genie DSP has 1000-PEC gain over the code using the current soft sensing method.

Note that the performance of polar codes shown in this section is optimized at each PEC—new polar codes are constructed with newly measured channel parameters for different PECs. However, switching codes at each different PEC is prohibitively expensive in practice. In the next section, we show that reconstructions of new codes are not needed to achieve such optimized performance.

8.3 Adaptive Decoding for Flash Memories

The channels of flash memories gradually degrade as PEC grows. Specifically, let the flash channel $\mathrm{W}(\alpha)$ be parameterized by PEC $\alpha \in \mathbb{N}$, $\mathrm{W}(\alpha) \succeq \mathrm{W}(\alpha')$ for any α, α' such that $\alpha \leqslant \alpha'$. To maintain error rates at the same low level, adaptive decoder is used in

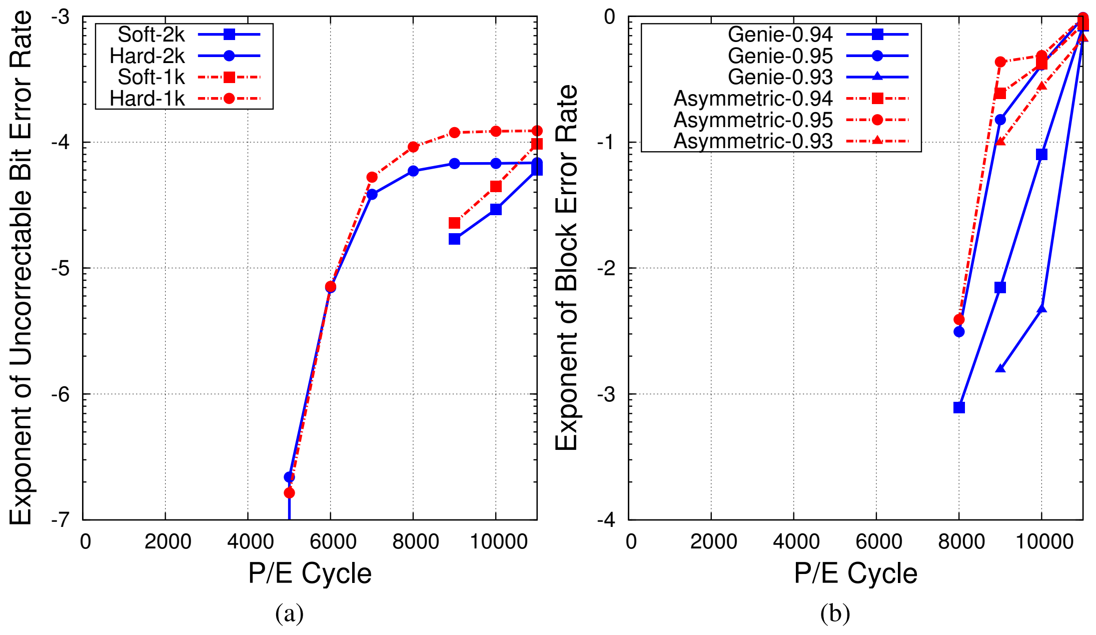


Figure 8.4: The performance of polar codes with (a) different block lengths as well as (b) realistic and genie DSPs.

practice where lower code rates are used when the channel becomes more noisy.

Definition 45. *Let $R_1 > R_2 > \cdots > R_{k-1}$ be $k-1$ code rates of some channel code C, and let $\alpha_1 < \alpha_2 < \cdots < \alpha_k$ be k selected PECs. For $i \in \{1, 2, \cdots, k-1\}$, an adaptive decoder of C is the decoder which*

1. *changes the rate of C to R_i at α_i.*
2. *uses rate R_i consistently for any $\alpha \in [\alpha_i, \alpha_{i+1})$.*

In this section, we show that polar codes is an excellent candidate for effective adaptive decoding in flash memories in the sense that the construction of new codes is not necessary through the lifetime of flash chips, and changing code rate only requires freezing additional input bits. The proof relies on the following lemma, which restates Corollary 1 from [20].

Lemma46. *Let F_W be the frozen set of the capacity-achieving polar codes for W. For any two channels W_i and W_j such that $W_i \succeq W_j$, the capacity achieving polar code for W_j can be obtained from the polar code for W_i by freezing additional input bits whose indices are in the set $F_{W_i} - F_{W_j}$.*

Consider an ideal adaptive polar decoder in Definition 45 with infinite code length, R_i being the capacity of $W(\alpha_i)$, and $\alpha_{i+1} = \alpha_i + 1$ for $i \in \{1, 2, \cdots, k-1\}$. The lemma above states that the ideal adaptive decoder can be realized by simply making additional input bits frozen when changing the rates at different PECs. In practice, adaptive decoders use finite block lengths, and it is prohibitively expensive to switch to a new code rate at each PEC. Therefore, we further consider a practical adaptive polar decoder with code rate R_i being smaller than the capacity of $W(\alpha_i)$, and $\alpha_{i+1} > \alpha_i + 1$ for $i \in \{1, 2, \cdots, k-1\}$. Let $W^{(1)}(\alpha), W^{(2)}(\alpha), \cdots, W^{(N)}(\alpha)$ be the N subchannels of the polar code for $W(\alpha)$. Let $\sigma_{W(\alpha)} = (x_1, x_2, \cdots, x_N)$ be the length-N permutation induced by the polarization order of the subchannels such that the sequence $P_e(W^{(x_1)}(\alpha)), P_e(W^{(x_2)}(\alpha)), \cdots, P_e(W^{(x_N)}(\alpha))$ is in ascending order where $P_e(\cdot)$ computes the theoretical decoding error rate of a channel.

Theorem47. *For any α, α' such that $\alpha \leqslant \alpha'$, and rate-R and rate-R' codes are used at α and α', respectively ($R > R'$), the polar code for $W(\alpha')$ can be obtained from the polar code for $W(\alpha')$ by further freezing the input bits in $F_{W(\alpha')} - F_{W(\alpha)}$ if*

$$\sigma_{W(\alpha)} = \sigma_{W(\alpha')}. \tag{8.2}$$

Proof: If (8.2) holds, $F_{W(\alpha)} = \{x_1, \cdots, x_{\lceil N(1-R) \rceil}\}$ and $F_{W(\alpha')} = \{x_1, \cdots, x_{\lceil N(1-R') \rceil}\}$. Therefore, to obtain code for $W(\alpha')$ only needs to frozen the additional input bits with indices in $\{x_{\lceil N(1-R) \rceil + 1}, x_{\lceil N(1-R) \rceil + 2}, \cdots, x_{\lceil N(1-R') \rceil}\}$ which is $F_{W(\alpha')} - F_{W(\alpha)}$. $\square$

The condition (8.2) is motivated by our experimental observations. Figure 8.5 shows the theoretical decoding error rates for each subchannel at different PECs for the polar codes used in Section 8.2-E. The figure suggests the error rates almost increase with PEC

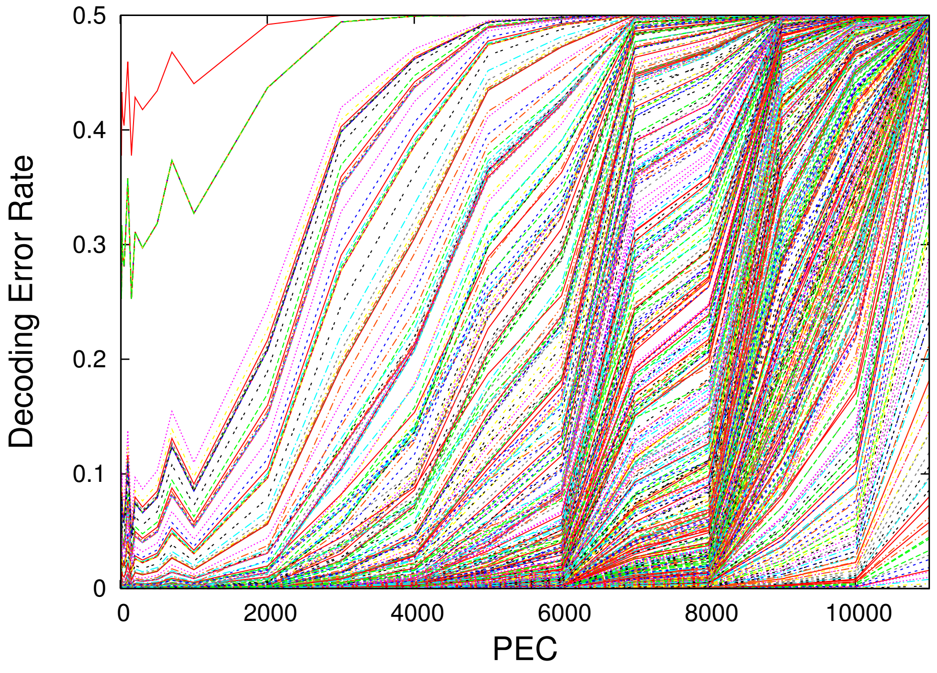
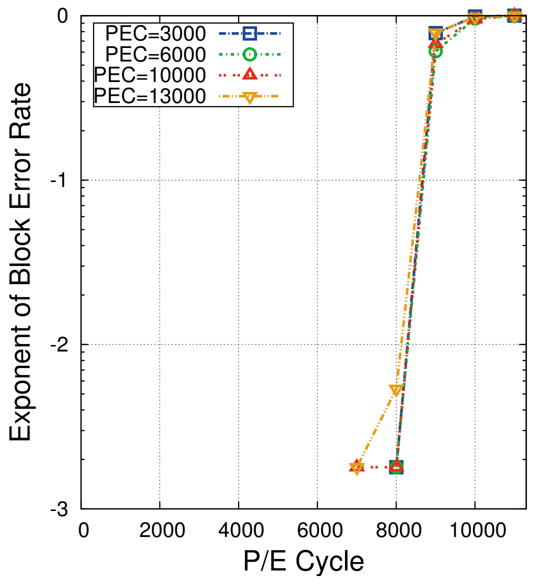


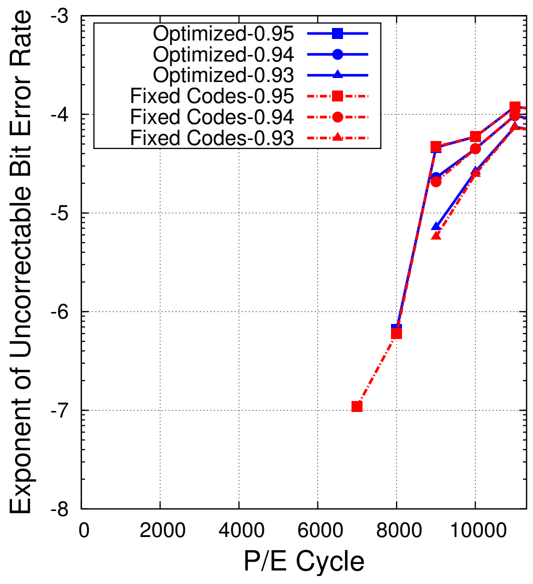
Figure 8.5: The theoretical decoding error rates of each subchannel at different PECs. Each curve denotes one subchannel.

(due to process variation, the rates do not increase monotonically between $0 - 10^3$ PECs), and that the order of polarization be well preserved. Assume (8.2) holds for any $\alpha, \alpha' \in [\alpha_1, \alpha_k]$. The next two corollaries state that the constructions of new codes can be avoided for practical adaptive decoders corresponding to 1) and 2) of Definition 45, respectively.

Corollary 48. *For $i \in \{1, 2, \cdots, k-1\}$, when the decoder changes the code rate R_i previously used at $\alpha_{i+1} - 1$ to R_{i+1} at α_{i+1}, it only needs to further make the input bits in $F_{w_{\alpha_{i+1}}} - F_{w_{\alpha_{i+1}-1}}$ frozen.*



(a) Average BERs of upper-odd pages



(b) Average UBERs over all pages

Figure 8.6: The performance of polar codes constructed at fixed PECs throughout the lifetime of the flash chips. Soft sensing is used.

Corollary 49. *For $i \in \{1, 2, \cdots, k-1\}$, given any two PECs $\alpha, \alpha' \in [\alpha_i, \alpha_{i+1})$, with the same code rate R_i the polar codes for $W(\alpha)$ and $W(\alpha')$ are equivalent.*

Corollary 49 states that no construction of new code is necessary for the PECs covered by the same code rate. Figure 8.6a shows the block error rates of four polar codes of rate-0.94 for the upper-odd pages constructed at PECs 3×10^3, 6×10^3, 10^4, and 1.3×10^4, respectively. Each code is tested through the whole lifetime of the flash chips. The results suggest the codes yield very similar decoding performance due to the polarization order preservation shown in Figure 8.5. Figure 8.6b compares the average UBERs of the codes constructed at 6000 PECs with the optimized performance yield by codes constructed at different PECs. The performance of the scheme without construction of new code closely approaches the optimized performance.

9. MULTI-PHASE SCRUBBING FOR PHASE-CHANGE MEMORIES

PCMs face important challenges on its reliability. One fundamental problem is caused by *resistance drift*: cell resistance drifts over time due to the structural relaxations in phase change materials. According to [8] [30], the resistance of a PCM cell increases with time t following:

$$\lg R_t = \lg R_{t_0} + v(\lg t - \lg t_0), \tag{9.1}$$

where R_t and R_{t_0} are the cell resistance at time t and t_0, respectively. The variable v is a drift parameter that varies from cell to cell, and increases with the initial resistance R_{t_0}. A common model assumes that $\lg R_{t_0}$ and v follows some probability distributions. Resistance drift shifts and broadens cell level distributions, introducing errors to data. An example of resistance drift is shown in Figure 9.1. (Note that the scheme presented in this paper is not restricted by the resistance drift model in Eq. 9.1. For other models such as the stochastic model presented in [22], similar results and analysis can be adapted.)

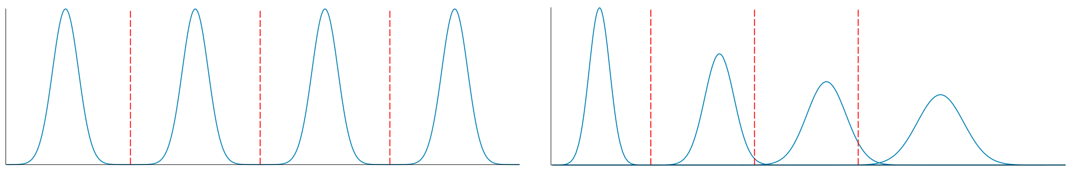

(a) Initial cell level distributions (b) Distributions after drift

Figure 9.1: Resistance drift within 4-level cells with fixed thresholds for levels. Horizontal axis: the logarithmic resistance. Solid curves: the resistance distributions of the cell levels. Dotted vertical lines: the resistance thresholds.

To prevent error accumulation caused by resistance drift, this paper studies an approach which can use the existing solutions and dynamically refresh PCM cell levels, namely memory scrubbing. Let the allowed memory operations be cell programming and reading. In each read, resistance is only measured once. It is not feasible to read the resistance of a cell multiple times in order to solve the values of R_{t_0} and v. This is because the memory system can not afford the space and delay cost by storing the analog resistance value for each cell. Therefore, we assume that the analog cell resistance measured during one read will be unavailable for the next read. The measured resistance is quantized into cell levels, or used for computing the input probabilities for soft decoding. The problem we consider in this paper is as follows.

Definition 50. *Let a sequence of n MLCs with q levels be initially programmed to store N bits at time t_0. Let cell resistance drifts following the model in Eq. (9.1). Design a scheme to allow the initially stored N bits be recovered at any given time $t > t_0$.*

Memory scrubbing has been widely used in dynamic random-access memories and flash memories. In a basic scheme, data are encoded with ECC, codewords are periodically decoded and written back to the cells. To prevent error accumulation, higher decoding frequency is needed for high rate ECCs. However, since the decoding frequency equals the scrubbing frequency in the basic scheme, increasing decoding frequency also brings more additional writes to the memory cells. The latter causes delay and degrades cell qualities.

In this paper, a new multi-phase memory scrubbing scheme is proposed for PCMs. Compared to the basic scheme, our scheme can increase decoding frequency without raising scrubbing frequency, and errors can be detected much earlier. In this scheme, cells storing an ECC codeword is partitioned into a few segments. The scrubbing proceeds in several phases. In each phase, one segment of cells is refreshed after decoding. The errors found during each decoding can also be saved as metadata to further improve the next decoding.

The rest of this paper is organized as follows. We construct the multi-phase scrubbing scheme in Section 9.1. We derive the achievable rates of the scheme in Section 9.2, and estimate its decoding error rates in Section 9.3. We discuss a further extension based on in-memory error detection in Section 9.4. The performance of our scheme is evaluated in Section 9.5.

9.1 Multi-phase Memory Scrubbing

In this section, We describe the multi-phase scrubbing scheme. Let $q = 2^m$ be the total number of levels in each cell, where $m \in \mathbb{N}$. An ECC with block length N is used for encoding K information bits $\mathbf{x} = (x_0, x_1, \cdots, x_{K-1}) \in \{0,1\}^K$ with the encoding function $\phi : \{0,1\}^K \to \{0,1\}^N$. Assume that $m|N$. Let the codeword $\mathbf{b} = (b_0, b_1, \cdots, b_{N-1}) \in \{0,1\}^N = \phi(\mathbf{x})$ be stored using n PCM cells denoted by $(C_0, C_1, \cdots, C_{n-1})$. Assume $m|N$, then $n = \frac{N}{m}$. For $i = 0, 1, \cdots, n-1$, the cell C_i is programmed to cell level $l_i \triangleq \pi^{-1}(b_{m\cdot i}, \cdots, b_{m\cdot(i+1)-1})$. Here $\pi : \{0,1\}^m \to \{0,1,\cdots,q-1\}$ is some binary-to-q-ary mapping, and let $\pi^{-1} : \{0,1,\cdots,q-1\} \to \{0,1\}^m$ be the inverse of π. Let $[T_{-1,t}, T_{0,t}, \cdots, T_{q-1,t}]$ be the thresholds used for quantizing cell resistance at time t, i.e., for $i \in \{0,1,\cdots,q-1\}$ a cell is at level i if and only if its logarithmic resistance $\lg R_t$ satisfies $T_{i-1,t} < \lg R_t < T_{i,t}$.

Construction 7. *Multi-phase scrubbing.*

Divide n cells equally into s segments denoted by $[S_0, S_1, \ldots, S_{s-1}]$ such that for $j \in \{0,1,\cdots,s-1\}$ segment S_j contains cells $\left(C_{j\cdot\lceil\frac{n}{s}\rceil}, \cdots, C_{\min\{(j+1)\cdot\lceil\frac{n}{s}\rceil-1, n-1\}}\right)$.

For $i = 0, 1, 2, \cdots$, at time $i \cdot t_m$, do the following:

1. *Read the cells, obtaining the codeword $\mathbf{b}' = (b'_0, b'_1, \cdots, b'_{N-1}) \in \{0,1\}^{N-1}$ from the cells with the resistance thresholds $[T_{-1,i\cdot t_m}, T_{0,i\cdot t_m}, \cdots, T_{q-1,i\cdot t_m}]$.*

2. *Decode* $\mathbf{b}'$. *Let* $(\hat{b}_0, \hat{b}_1, \cdots, \hat{b}_{N-1}) = \phi^{-1}(\mathbf{b}')$. *Upon decoding failure, stop and report the failure.*

3. *Let* $j = i \mod s$, *refresh* S_j *by programming cells* $(C_{j\cdot\lceil\frac{n}{s}\rceil}, \cdots, C_{\min\{(j+1)\cdot\lceil\frac{n}{s}\rceil-1, n-1\}})$ *to the levels* $(\hat{l}_{j\cdot\lceil\frac{n}{s}\rceil}, \cdots, \hat{l}_{\min\{(j+1)\cdot\lceil\frac{n}{s}\rceil-1,n-1\}})$ *where* $\hat{l}_y \triangleq \pi^{-1}((\hat{b}_{m\cdot y}, \cdots, \hat{b}_{m\cdot(y+1)-1}))$ *for* $y = j \cdot \lceil\frac{n}{s}\rceil, \cdots, \min\{(j+1)\cdot\lceil\frac{n}{s}\rceil - 1, n-1\}$.

Remark 51. *The multi-phase scheme has scrubbing frequency* $\frac{1}{s\cdot t_m}$ *and decoding frequency* $\frac{1}{t_m}$.

Remark 52. *Let* $g(i,j) \triangleq (s - (j-i) \mod s)\cdot t_m$ *if* $i \neq j$, *and* $g(i,j) \triangleq s\cdot t_m$ *otherwise. At the moment when we are about to decode* $\mathbf{b}'$ *to refresh* S_i, *the cells in* S_j *have drifted for time* $g(i,j)$.

In Construction 7, the errors found in the cell segments which are not being scrubbed are immediately forgotten. We propose an extension to store such error information as *metadata*. As metadata are also effected by channel noise, and shall not occupy too much storage space, metadata need to be compressed and protected with ECC.

In the following, we assume that segment S_i is being scrubbed, and that the noisy codeword $\mathbf{b}'$ corresponds to a sequence of cell levels $\mathbf{l}' = (l'_0, l'_1, \cdots, l'_{n-1}) \in \{0, 1, \cdots, q-1\}^n$. Let the estimated codeword $(\hat{b}_0, \hat{b}_1, \cdots, \hat{b}_{N-1}) = \phi^{-1}(\mathbf{b}')$ given by ECC decoder correspond to the estimated cell levels $(\hat{l}_0, \hat{l}_1, \cdots, \hat{l}_{n-1}) \in \{0, 1, \cdots, q-1\}^n$. Let $\psi(\cdot)$ be the encoding function of some source code, and let $\phi_{\text{meta}}(\cdot)$ be the encoding function of some ECC for protecting the metadata.

Construction 8. *Generating metadata.*
Let $\tilde{\mathbf{l}} \triangleq (\tilde{l}_0, \tilde{l}_1, \cdots, \tilde{l}_{\lceil\frac{n}{s}\rceil-1}) \in \{0, 1, \cdots, q\}^{\lceil\frac{n}{s}\rceil}$ *be a sequence of* $(q+1)$*-ary symbols. Let* $z = (i+1) \mod s$ *be the index of the segment to be scrubbed in the next, and let*

$A \triangleq \left\{ j \mid z \cdot \lceil \frac{n}{s} \rceil \leqslant j \leqslant \min\{(z+1)\cdot\lceil \frac{n}{s} \rceil - 1, n-1\}, \hat{l}_j \neq l'_j \right\}$ *be the set of indices of the erroneous cells in segment* S_z.

- *For* $j \in A$, *let* $\tilde{l}_{j-z\cdot\lceil \frac{n}{s} \rceil} = \hat{l}_j$.
- *For* $j \in \left\{ z \cdot \lceil \frac{n}{s} \rceil, \cdots, \min\{(z+1)\cdot\lceil \frac{n}{s} \rceil - 1, n-1\} \right\} - A$, *let* $\tilde{l}_{j-z\cdot\lceil \frac{n}{s} \rceil} = q$.

Store the codeword $\mathbf{m} = \phi_m(\psi(\tilde{\mathbf{l}}))$ *using additional cells.*

Remark 53. *The cells for storing metadata are reprogrammed every time* t_m.

When scrubbing a segment, metadata are read first and used to correct the codeword. The codeword is then passed to ECC decoder.

Construction 9. *Error correction using metadata.*
If metadata are used, do the following before decoding $\mathbf{b}'$ *(step 2 of Construction 1):*

1. *Read from metadata, obtaining a noisy codeword* $\mathbf{m}'$.
2. *Compute the estimated sequence*

$$\hat{\tilde{\mathbf{l}}} = \psi^{-1}(\phi_m^{-1}(\mathbf{m}')) = (\hat{\tilde{l}}_0, \hat{\tilde{l}}_1, \cdots, \hat{\tilde{l}}_{\lceil \frac{n}{s} \rceil - 1}) \in \{0, 1, \cdots, q\}^{\lceil \frac{n}{s} \rceil}.$$

   Upon decoding failure, skip the rest of the steps.

3. *Correct the part of* $\mathbf{b}'$ *stored in segment* S_i. *For* $j \in \{0, 1, \cdots, \lceil \frac{n}{s} \rceil - 1\}$, *if* $\hat{\tilde{l}}_j \neq q$, *let* $(b'_{m(i\cdot\lceil \frac{n}{s} \rceil + j)}, \cdots, b'_{m(i\cdot\lceil \frac{n}{s} \rceil + j + 1) - 1}) = \pi(\hat{\tilde{l}}_j)$.

Example 54. *Let* $N = 12$, $K = 8$, $q = 4$, $s = 3$. *Therefore,* $n = 6$ *and segment* S_0 *has cells* (C_0, C_1), S_1 *has cells* (C_2, C_3), *and* S_2 *has cells* (C_4, C_5). *For simplicity, we assume metadata are noiseless and only show the values of* $\tilde{\mathbf{l}}$. *As shown in Figure 9.2, let the initial cell states correspond to a binary* $(12, 8)$*-shortened Hamming codeword that*

Initial States	2	0	1	3	1	3	
			$\cdots$				$\tilde{\mathbf{l}}$
Scrub S_0	2	0	$\underline{2}$	3	1	3	$(1, 4)$
Scrub S_1	2	0	1	3	$\underline{3}$	3	$(1, 4)$
Scrub S_2	2	0	1	3	1	3	$(4, 4)$
			$\cdots$				

Figure 9.2: An example for illustrating multi-phase scrubbing.

corrects 1 *error. Suppose that at some moment when we scrub* S_0*, ECC decoding finds that cell* C_2 *is erroneous (marked by underline), the correct level* 1 *of* C_2 *is then recorded in* $\tilde{l}_0$*. As* C_3 *has no error,* $\tilde{l}_1 = q = 4$*. Next, we scrub* S_1*, assume* C_4 *becomes faulty (in addition to* C_2*). Before decoding the noisy codeword, we first correct the bits stored by* C_2 *using the metadata. We then use decoding to recover the level of* C_4*, updating* $\tilde{l}_0 = 1$*. We then scrub* S_2*. Suppose that only* C_4 *is faulty,* $\tilde{l}_0$ *is used to correct the bits of* C_4*. Since the decoding finds no error, the values of* $\tilde{l}_0$ *and* $\tilde{l}_1$ *are set to* 4*.*

The multi-phase scrubbing can use any existing ECCs. Since the resistance thresholds in our scheme is parametrized by time t, our scheme naturally supports the use of dynamic resistance thresholds [78] [88]. Besides the error logging mechanism proposed in Constructions 8 and 9, our scheme can be adapted to work with more advanced error logging schemes such as [64] for better performance.

9.2 Code Rate Analysis

In this section, we analyze the achievable rates of the multi-phase scrubbing scheme using metadata when $n \to \infty$.

9.2.1 Channel Model

We first define the channel model. To simplify the analysis, we assume $s|n$. For $i \in \{0, 1, \cdots, q-1\}$, we use $\lg R_{i,t}$ to denote the logarithmic resistance of a cell at time t which is initially programmed to cell level i at time t_0. Let the initial logarithmic cell resistance $\lg R_{i,t_0}$ and the drift exponent v each follow a Gaussian distribution

$$\lg R_{i,t_0} \sim \mathcal{N}(\mu_{\lg R_{i,t_0}}, \sigma^2_{\lg R_{i,t_0}}), v \sim \mathcal{N}(\mu_v, \sigma_v^2).$$

Then the logarithmic resistance $\lg R_{i,t} \sim \mathcal{N}(\mu_{\lg R_{i,t}}, \sigma^2_{\lg R_{i,t}})$ where

$$\mu_{\lg R_{i,t}} = \mu_{\lg R_{i,t_0}} + \mu_v(\lg t - \lg t_0),$$

$$\sigma^2_{\lg R_{i,t}} = \sigma^2_{\lg R_{i,t_0}} + \sigma_v^2(\lg t - \lg t_0)^2.$$

We use $\Pr\{l_t = j \mid l_{t_0} = i\} \triangleq \Pr\{T_{j-1,t} < \lg R_{i,t} \leqslant T_{j,t}\}$ to denote the probability that a cell with initial level i at time t_0 drifts to level j at time t, and we have

$$\Pr\{l_t = j \mid l_{t_0} = i\} = \int_{T_{j-1,t}}^{T_{j,t}} f(x)\,\mathrm{d}x,$$

where function $f(\cdot)$ is the probability density function of the Gaussian distribution

$$\mathcal{N}(\mu_{\lg R_{i,t}}, \sigma^2_{\lg R_{i,t}}).$$

Assume each bit in a codeword is *i.i.d.*. For $i \in \{0, 1, \cdots, m-1\}$, let β_i be the probability that the i-th bit stored in a cell is 1. The probability that a cell is initially programmed to level $j \in \{0, 1, \cdots, q-1\}$ to be

$$\Pr\{l_{t_0} = j\} \triangleq \Pr\{(b_0, b_1, \cdots, b_{m-1}) = \pi(j)\}$$
$$= \prod\nolimits_{x \in \{0,1,\cdots,m-1\}} \beta_x^{b_x}(1-\beta_x)^{1-b_x}.$$

9.2.2 Code Rates

We now compute the achievable rates. Assume we are at the moment when segment S_a is about to be scrubbed. Given a vector $v = (v_0, v_1, \cdots, v_{k-1}) \in \{0,1\}^k$, define its support as $support(v) \triangleq \{i \mid i \in \{0, 1, \cdots, k-1\}, v_i = 1\}$. We define the *cross* of i as

$$cross(i) \triangleq \{j | j \in \{0, 1, \cdots, q-1\}, i \in support(\pi(j))\}$$

which computes the set of cell levels whose corresponding binary representation has 1 in the i-th bit.

Example55. *Let* $m = 3$*, and let* π *follow the basic binary mapping, i.e.,* $0 \to (0,0,0)$, $1 \to (0,0,1)$, $2 \to (0,1,0)$, $\cdots$, $7 \to (1,1,1)$. *Then* $cross(0) = \{1,3,5,7\}$, $cross(1) = \{2,3,6,7\}$*, and* $cross(2) = \{4,5,6,7\}$.

Let $b_{i,j}$ be the i-th bit stored in a cell of segment S_j, and let $e_{i,j}$ denote the bit error received by $b_{i,j}$. For $j \in \{0, 1, \cdots, s-1\} - \{a\}$, the cross-over probabilities for $b_{i,j}$ are

$$\Pr\{e_{i,j} = 1 \mid b_{i,j} = 0\} = \sum\nolimits_{k \in \overline{cross}(i)} \sum\nolimits_{k' \in cross(i)}$$
$$\Pr\{l_{t_0} = k \mid b_{i,j} = 0\} \Pr\{l_{g(a,j)} = k' \mid l_{t_0} = k\},$$

$$\Pr\{e_{i,j} = 1 \mid b_{i,j} = 1\} = \sum\nolimits_{k \in cross(i)} \sum\nolimits_{k' \in \overline{cross}(i)}$$
$$\Pr\{l_{t_0} = k \mid b_{i,j} = 1\} \Pr\{l_{g(a,j)} = k' \mid l_{t_0} = k\}.$$

The bits stored in S_a is corrected with metadata first before decoding, for $j = a$ the cross-over probabilities are

$$\Pr\{e_{i,a} = 1 \mid b_{i,a} = 0\} = \sum\nolimits_{k \in \overline{cross}(i)} \sum\nolimits_{k' \in cross(i)} \\ \Pr\{l_{t_0} = k \mid b_{i,a} = 0\} \Pr\{l_{st_m} = k', l_{(s-1)t_m} = k \mid l_{t_0} = k\},$$

$$\Pr\{e_{i,a} = 1 \mid b_{i,a} = 1\} = \sum\nolimits_{k \in cross(i)} \sum\nolimits_{k' \in \overline{cross}(i)} \\ \Pr\{l_{t_0} = k \mid b_{i,a} = 1\} \Pr\{l_{st_m} = k', l_{(s-1)t_m} = k \mid l_{t_0} = k\}.$$

Here $\Pr\{l_{s \cdot t_m} = j, l_{(s-1) \cdot t_m} = i \mid l_{t_0} = i\}$ is the probability that the level of a cell is correct at time $(s-1) \cdot t_m$ and turns faulty at time $s \cdot t_m$, and for $s = 1$, the probability becomes $\Pr\{l_{t_m} = j \mid l_{t_0} = i\}$. The rate $R_{i,j}$ of the i-th bit of a cell in segment S_j is

$$\begin{aligned} R_{i,j} =& I(X;Y) \\ =& \mathrm{H}((1-\beta_i)\Pr\{e_{i,j} = 0 \mid b_{i,j} = 0\} + \beta_i \Pr\{e_{i,j} = 1 \mid b_{i,j} = 1\}) \\ &-(1-\beta_i)\mathrm{H}(\Pr\{e_{i,j} = 1 \mid b_{i,j} = 0\}) - \beta_i \mathrm{H}(\Pr\{e_{i,j} = 1 \mid b_{i,j} = 1\}). \end{aligned}$$

where $\mathrm{H}(\cdot)$ is the binary entropy function.

Lemma56. *The ECC for the information bits has rate*

$$\frac{1}{s}\sum_{j=0}^{s-1}\sum_{i=0}^{m-1} R_{i,j} \text{ bits/cell.}$$

When generating metadata, the $(q+1)$-ary sequence $\tilde{\mathrm{l}}$ are first compressed. The achievable compression rate follows Shannon's lossless source coding theorem. Let p_i be the probability that a symbol $i \in \{0, 1, \cdots, q\}$ appears in $\tilde{\mathrm{l}}$. As metadata store error information for the next segment to be scrubbed, and the symbol $i \in \{0, 1, \cdots, q-1\}$ appears if and only if a cell in the next segment is initially at level i and becomes erroneous before the decoding of the current scrubbing iteration, we have for $i \in \{0, 1, \cdots, q-1\}$, $p_i = \Pr\{l_{t_0} = i\} \sum_{j=0, j \neq i}^{q-1} \Pr\{l_{(s-1)\cdot t_m} = j \mid l_{t_0} = i\}$, and $p_q = 1 - \sum_{j=0}^{q-1} p_i$.

Lemma 57. *The sequence* $\tilde{\mathrm{l}}$ *is compressed to* $\frac{n}{s} \sum_{i=0}^{q} p_i \log_2 \frac{1}{p_i}$ *bits.*

Since we assume that the compression is optimal, each bit in the source codeword has equivalent probability of being 0 and 1. The source codeword representing $\tilde{\mathrm{l}}$ needs to be encoded with ECC. Let $b_{\text{meta},i}$ be the i-th bit of a cell used for storing metadata. Let $e_{\text{meta},i}$ be the error received by $b_{\text{meta},i}$. As metadata experiences the same channel noise as the information bits do, the cross-over probabilities $\Pr\{e_{\text{meta},i} = 1 \mid b_{\text{meta},i} = 0\}$ and $\Pr\{e_{\text{meta},i} = 1 \mid b_{\text{meta},i} = 1\}$ can be derived accordingly. The rate of the i-th bit of a cell for storing metadata is

$$\begin{aligned} R_{\text{meta},i} =& I(X;Y) \\ =& \mathrm{H}(\frac{1}{2}\Pr\{e_{\text{meta},i} = 0 \mid b_{\text{meta},i} = 0\} \\ &+ \frac{1}{2}\Pr\{e_{\text{meta},i} = 1 \mid b_{\text{meta},i} = 1\}) \\ &- \frac{1}{2}\mathrm{H}(\Pr\{e_{\text{meta},i} = 1 \mid b_{\text{meta},i} = 0\}) \\ &- \frac{1}{2}\mathrm{H}(\Pr\{e_{\text{meta},i} = 1 \mid b_{\text{meta},i} = 1\}). \end{aligned}$$

Lemma 58. *The ECC for protecting the metadata has rate*

$$\sum_{i=0}^{m-1} R_{\text{meta},i} \textit{ bits/cell.}$$

Lemma 59. *The total number of additional PCM cells used for storing metadata is*

$$\frac{n \sum_{i=0}^{q} p_i \log_2 \frac{1}{p_i}}{s \sum_{i=0}^{m-1} R_{\text{meta},i}}.$$

Theorem 60. *The achievable rate of the multi-phase scrubbing scheme using metadata is*

$$\max_{\beta_0,\beta_1,\cdots,\beta_{m-1}\in[0,1]} \frac{\sum_{j=0}^{s-1}\sum_{i=0}^{m-1} R_{i,j} \cdot \sum_{i=0}^{m-1} R_{\text{meta},i}}{s\sum_{i=0}^{m-1} R_{\text{meta},i} + \sum_{i=0}^{q} p_i \log_2 \frac{1}{p_i}} \textit{ bits/cell.}$$

9.3 Decoding Error Rate Analysis

In practice, the performance of memory scrubbing depends on the specific ECCs used in the scheme. In this section, we analyze the decoding error rates that can be achieved by our scheme using ECCs with finite block lengths and known correction capabilities.

The expected number of errors in a cell $\mathbb{E}(i,t)$ is a function of the initial level i and the time t elapsed since the initial level was programmed:

$$\mathbb{E}(i,t) = \sum_{j=0,j\neq i}^{q-1} h(\pi(i),\pi(j)) \Pr(l_t = j | l_{t_0} = i),$$

where $h(\cdot,\cdot)$ computes the Hamming distance between two binary strings. The expected bit error rate (BER) for a cell programmed time t ago is estimated as:

$$\mathrm{B}(t) = \frac{1}{qm}\sum_{i=0}^{q-1}\mathbb{E}(i,t).$$

9.3.1 Decoding Error Rates without Metadata

Let an (N,K,d,c)-ECC which has block length N, information bit length K, minimum distance d, and corrects up to c errors be used for encoding the information bits. A block error occurs when $e > c$. Cells belonging to different codeword segments drift for different amount of time before decoding. The average BER $\overline{B}$ immediately before each decoding is given by $\frac{1}{s}\sum_{i=1}^{s} \mathrm{B}(i \cdot t_m)$. The decoding block error rate of the multi-phase scheme is:

$$B_{\text{multi}} = \sum_{i=c+1}^{N} \binom{N}{i} \overline{B}^{i} (1 - \overline{B})^{n-i}.$$

9.3.2 Decoding Error Rates with Metadata

Let the information bits be encoded with an (N_1, K_1, d_1, c_1)-ECC C_1, and the metadata be encoded with an (N_2, K_2, d_2, c_2)-ECC C_2. Let e_1 and e_2 respectively be the number of errors in the codewords of C_1 and C_2 before decoding. A block error occurs when: (1) $e_2 \leqslant c_2$ and $e_1 > c_1 + \Delta$: after correcting Δ errors in the current segment with metadata, the number of errors left in the codeword of C_1 still exceeds C_1's correction capability. Here the Δ errors are found when the cells in the current segment has drifted for time $(s-1) \cdot t_m$, and we have $\Delta = \frac{N_1}{s} \cdot \mathrm{B}((s-1) \cdot t_m)$. (2) $e_2 > c_2$ and $e_1 > c_1$: metadata are not used due to the decoding failure of C_2. The number of errors in the codeword of C_1 exceeds the correction capability. The decoding error rate is calculated from the two cases above:

$$B_{\text{meta}} = \mathrm{P}(e_2 > c_2)\, \mathrm{P}(e_1 > c_1) + \mathrm{P}(e_2 \leqslant c_2)\, \mathrm{P}(e_1 > c_1 + \Delta),$$

where the probabilities above are computed as:

$$\mathrm{P}(e_2 > c_2) = \sum_{i=c_2+1}^{N_2} \binom{N_2}{i} \mathrm{B}(t_m)^i (1 - \mathrm{B}(t_m))^{N_2 - i},$$

$$\mathrm{P}(e_1 > c_1) = \sum_{i=c_1+1}^{N_1} \binom{N_1}{i} \overline{B}^{i} (1 - \overline{B})^{N_1 - i},$$

$$\mathrm{P}(e_1 > c_1 + \Delta) = \sum_{i=c_1+\Delta+1}^{n_1} \binom{N_1}{i} \overline{B}^{i} (1 - \overline{B})^{N_1 - i}.$$

9.4 In-memory Error Detection

During memory scrubbing, an ECC codeword needs to be read from cells into the processor of memory controller. The latter decodes the codeword to correct errors. To prevent error accumulation, one way is to use an ECC which corrects a small number of errors, and decode the codeword in a high frequency. However, high decoding frequency brings a large number of memory reads which introduces significant delay. Another option is to use ECC which can correct many errors to reduce decoding frequency. However, this approach introduces more parity check bits, and decoding becomes more expensive. To obtain high decoding frequency without introducing much delay in memory controller, we extend memory scrubbing by adding an in-memory circuit which detects a few errors. Since the error detection is done in circuit, detection can operate at a much higher frequency than memory scrubbing does. Once a few errors are detected by the circuit, memory scrubbing scheme is triggered to correct all the errors and refresh the cell levels. In rare cases where many errors occur at the same time, in-memory error detection fails, and we will wait for memory scrubbing cycles to correct all the errors.

One way to construct the in-memory error detection circuit is studied in [79]. A circuit can be built to detect up to t errors by checking a set of parity check equations. The selected parity check equations form a subset of all the parity checks given by the parity check matrix of the ECC for encoding the data. The construction uses existing ECC therefore no additional redundancy is added. However, the problem for picking the smallest parity check set for detecting t errors can be reduced to a variation of the standard set cover problem. Due to the NP-hardness of the problem, the construction uses a greedy approximation algorithm [43] whose complexity grows exponentially with t. A more efficient construction method is currently being studied and will be included in our future work. The other way to construct the circuit is to use concatenated error correction codes. Different from the construction in [79], this approach has efficient construction algorithms, but introduces additional redundancy. For instance, information bits can be encoded by cyclic redundancy check (CRC) [58] first, then encoded by a full-strength ECC such as BCH codes or LDPC codes. CRCs have various error detection capabilities depending on the number of check bits they use, and have very efficient circuit implementation.

9.5 Performance Evaluation

In this section, we evaluate the performance of the multi-phase scrubbing scheme. We compare the performance of three schemes: (1) multi-phase scheme with metadata. (2) Multi-phase scheme without metadata. (3) A basic memory scheme, which periodically decodes the codewords, and then refreshes all the cells.

We always assume that all the schemes have the same scrubbing frequency, namely the scrubbing frequency and the decoding frequency of the basic scheme is $\frac{1}{s \cdot t_m}$. Let $q = 4$, and let cells have the configurations in Table 9.1. Cell levels are determined using fixed resistance thresholds. The threshold between two adjacent levels is set to the middle of the means of their initial logarithmic resistance.

Table 9.1: The configurations of PCM cells used for evaluations.

Level i	**Bits** $\pi(i)$	$\mu_{\lg R_{i,t_0}}$	$\sigma_{\lg R_{i,t_0}}$	μ_v	σ_v
0	00	3.00	0.13	0.005	0.020
1	01	4.00	0.13	0.080	0.032
2	10	5.00	0.13	0.090	0.036
3	11	6.00	0.13	0.100	0.040

9.5.1 Code Rates

We first show some numerical results of the achievable code rates of the three schemes in Figure 9.3 (The achievable rates of the multi-phase scheme without metadata and the basic scheme can be computed correspondingly). We assume that $\beta_i = \frac{1}{2}$ for $i \in \{0, 1, \cdots, m-1\}$. Higher rates can be achieved for smaller scrubbing period. This is because smaller scrubbing period gives better channel quality. The multi-phase schemes (with or without metadata) achieve higher rates than the basic scheme does. By using metadata increases the rate of the multi-phase scheme further. The performance of multi-phase schemes degrades nicely when the number of segments changes from 16 to 8.

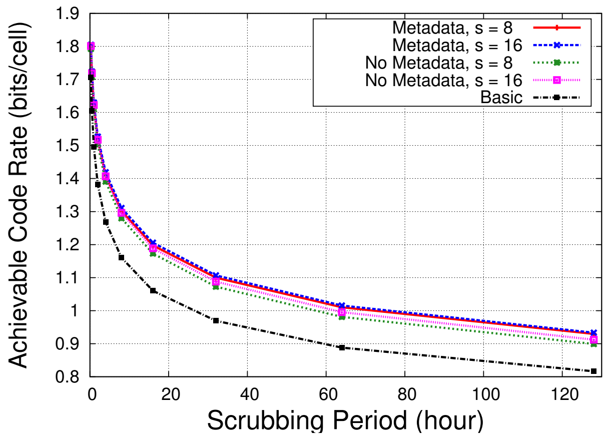


Figure 9.3: The code rates of the three scrubbing schemes given different s.

9.5.2 *Estimated Decoding Bit Error Rates*

We evaluate the decoding bit error rates of the multi-phase schemes with or without metadata as well as the basic schemes. Assume that the information bits are encoded with a $(4095, 2081, 341, 170)$-BCH code. Let $q = 4$, $s = 4$. Assume a decoded codeword is still close to the original codeword when error occurs, the decoding BER of each scheme is estimated by $\frac{d}{n} \cdot B$, where $B = B_{\text{multi}}$ for our scheme without metadata, $B = B_{\text{meta}}$ for our scheme using metadata, and $B = \sum_{i=c+1}^{N} \binom{N}{i} \mathrm{B}(s \cdot t_m)^i (1 - \mathrm{B}(s \cdot t_m))^{N-i}$. For the extended multi-phase scheme, we compute the upper bound of the decoding BER

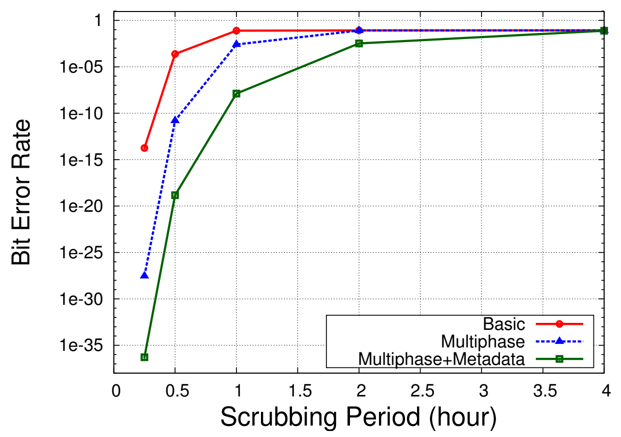


Figure 9.4: The decoding BERs of each scheme with different scrubbing periods.

by assuming the decoding of the metadata always succeeds. The upper bound can be achieved when t_m is small. Each scheme is evaluated using five scrubbing time periods (in hour units): $0.25\,\mathrm{h}$, $0.5\,\mathrm{h}$, $1\,\mathrm{h}$, $2\,\mathrm{h}$ and $4\,\mathrm{h}$. Figure 9.4 shows the BERs for each scheme evaluated at different scrubbing periods. The multi-phase scheme outperforms the basic scheme when the scrubbing period is less than $2\,\mathrm{h}$. When the scrubbing period is larger than $2\,\mathrm{h}$, errors exceed the correction capabilities of all the schemes. The BERs of the three schemes are very similar.

9.5.3 Simulation Results

We conducted simulations to evaluate the experimental performance of our schemes. For each scheme, we measured the average BER immediately before each scrubbing to compare the capability of each scheme on preventing error accumulation. The information bits are encoded using a rate-0.94 LDPC code with block length 4376 constructed following [51]. The decoder uses the sum-product algorithm with iteration threshold 32. The soft information is computed using the method in [78]. The metadata were compressed using Huffman coding and were encoded with a rate-0.87 LDPC code. For LDPC soft decoding. Assume $s = 4$. The simulated data retention period 512 days, and let

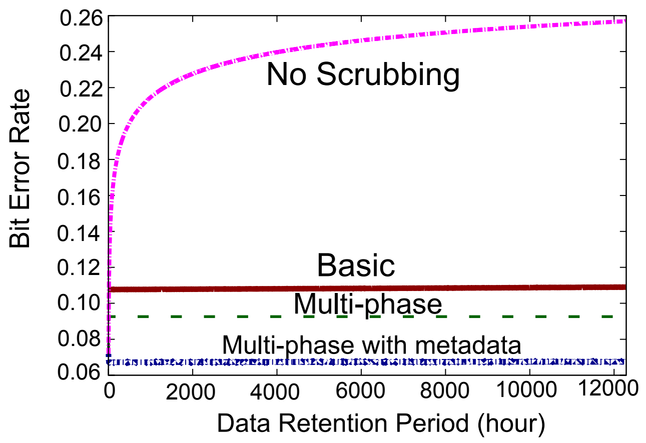


Figure 9.5: The BERs before every decoding during retention given $t_m = 3\,\mathrm{h}$.

$t_m = 3\,\mathrm{h}$. The BERs before scrubbings are shown in Figure 9.5. Without using scrubbing, the raw bit error rate grows quickly in the early stage of the whole retention period. The multi-phase scheme without using metadata keeps BER much lower than the basic scheme does. By using metadata further lowers the BERs.

10. EMULATING RANK MODULATION CODES IN FLASH MEMORIES

The theoretical study of rank modulation codes as well as their potential benefits were well established in the literature. The main advantage of rank modulation is that charge leakage is tolerated well with permutations. Since charge leakage behaves similarly in spatially close cells, the order of the cell levels is not likely to change. In comparison, when the information is represented by a quantization of absolute values, the leakage is more likely to introduce errors.

However, no rigorous study has been carried to explore the challenges and practical advantages in their physical implementation. The main objective of this work is to close on this theoretical and practical gap and pioneeringly implement rank modulation codes in flash memory chips. Our point of departure starts with a test board and different flash memory chips. In this chapter, we discuss the ongoing work on experimenting rank modulation in flash memories. We introduce our experimental platforms, methodologies, ongoing experiments, early results, and milestones that we plan to reach in the future.

10.1 Experimental Platform

10.1.1 Flash Test Board

Our main experimental tool is the flash test board shown in Figure 10.1. The board is designed by the group of Professor Edwin Kan at Cornell University [74, 75]. We obtained the board under the help of Dr. Yanjun Ma from Intellectual Ventures. The board contains four major components: (1) a socket for holding NAND flash chips (in TSOP-48 packaging), (2) an ARM7 (LPC2148) microcontroller for operating the flash chip (e.g. reading and writing), (3) an USB interface for power supply and data communication with host (e.g. PC), and (4) a serial programmer for receiving compiled executable from host and programming the microcontroller. The board serves as our initial setup for emulating and characterizing rank modulation in NAND flash memories as discussed later.

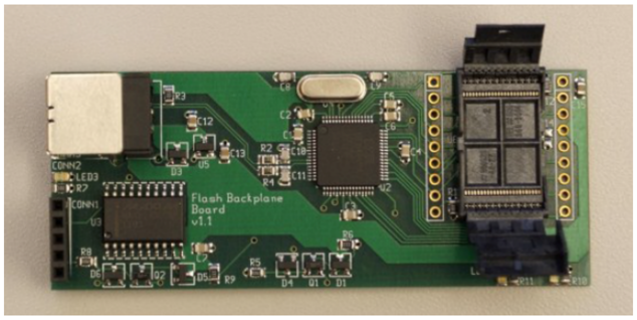


Figure 10.1: The flash test board.

10.1.2 NAND Flash Chips

The flash chips being used in our experiments are listed in Table 10.1. These chips are existing products that are publicly available on the market, supporting the Open NAND Flash Interface (ONFI) [32]. Chips with both single-level cells (SLCs) and multi-level cells (MLCs) were used. An SLC has two levels 1 and 0, it stores 1 bit per cell, and is read by comparing the threshold voltage with one reference threshold voltage. An MLC stores 2 bits per cell, and has four levels. Each bit stored in MLC is independently read using different reference threshold voltages.

Table 10.1: The NAND flash chips used in our experiments.

Chip Name	**Type**	**ONFI version**	**Size**
Micron MT29F4G08ABADAWP	34nm SLC	1.0	8GB
Micron MT29F64G08CFACA	25nm MLC	2.2	64GB
Hynix H27UAG8T2BTR-BC	32nm MLC	2.2	16GB

10.1.3 Support from Flash Chip Manufacturers

In addition to the flash test board, we have also initiated our efforts to collaborate with chip manufacturers. One of the leading flash chip manufacturers Toshiba has agreed to provide their latest generation of multi-level flash memory chips as well as the information regarding the advanced testing commands for the chips. With such, the experimental performance of rank modulation scheme can be better characterized, and the implementation complexity will be greatly reduced. Besides Toshiba, rank modulation scheme will be experimented with the chips from other major chip manufacturers which include Micron, Hynix, SanDisk, as well as Samsung. In the future, we plan to obtain the support from these manufacturers as well so that the benefits of rank modulation will be demonstrated for general flash chips.

10.2 Emulating Rank Modulation

As a first step, we would like to emulate rank modulation in existing NAND flash without modifying the chips. The emulation will allow us to conduct various kinds of characterizations on the practical performance of rank modulation scheme.

The emulation requires designing algorithms for writing and reading rank modulation codewords (i.e. permutations). To write a permutation, we need to program each cell in the same group to different intermediate state. To read a permutation, the programmed intermediate states of the cells need to be compared to assign ranks. In our experiments, such operations are based on an important programming technique referred as partial programming (PP). The capability of PP was discovered in [74]. In general, PP is a timed programming operation implemented by using the programming command followed by an abort command. To partially program a cell, a time threshold is given, the programming command is applied for a certain amount of time specified by user, after which the operation is aborted, and the cell will stay in an intermediate state. To compare the inter-

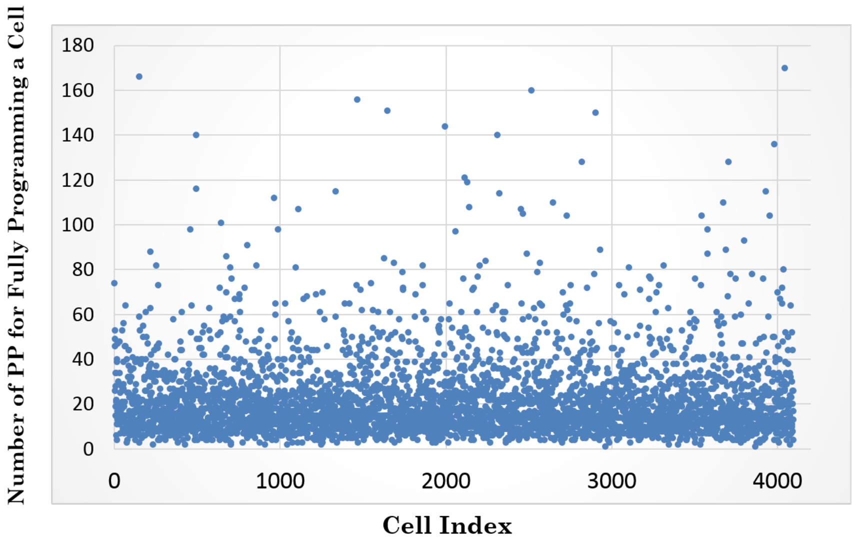


Figure 10.2: The first 4096 cells of a page in a Micron 34nm SLC flash chip with PP duration = 27 microseconds

mediate states, we use the number of PPs needed for each cell to reach level 0.

One of the challenges for using PP for rank modulation is due to the process variation. Let us refer to the number of PP needed to program an erased cell to level 0 as the *cell resolution*. Due to the variation in manufacturing, different cells may have significantly different resolutions. Figure 10.2 shows the number of PPs taken to program a cell from level 1 to level 0 for the first 4096 cells in an SLC flash memory page. The results suggest that most of the cells need a small number of PPs to reach level 0, while some cells need a significantly larger number of PPs to do the same. Figure 10.3 shows the number of PPs needed to program an erased cell to 0 using PPs with different time thresholds. Naturally, the number of PPs needed to program a cell decreases when the PP duration increases. When PP duration is very small, a much larger number of PPs is needed to program a cell; when time threshold is very large, PP has a similar behavior as a full programming command does. The results suggest that we shall select the PP duration to some time threshold in between.

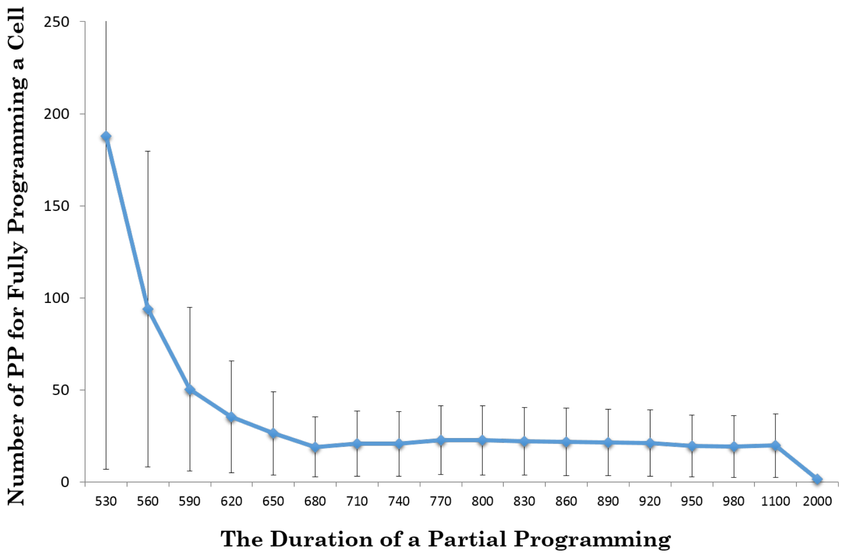


Figure 10.3: The first 4096 cells of a page in a Micron 34nm SLC flash chip with PP duration = 27 microseconds

We propose two schemes for emulating rank modulation that are robust under process variation. Namely, fine tuning based programming (FTP) and hybrid programming (HP). In FTP, we first measure the average resolution of each cell to be programmed. Given the rank of a cell, the number of PPs needed is determined according to the resolution. For instance, let there be three cells c_1, c_2, c_3 that needs to store the length-3 permutation $(1, 2, 3)$. Assume the average resolution of each cell is 3, 6 and 5, respectively. To store the permutation, we can apply 3 PPs to c_1, 2 PPs to c_2, and leave c_3 untouched. To read the stored permutation, we continue applying PP to each cell and record the number of PPs still needed for each cell to reach level 0. Together with the measured average resolutions, we can estimate the previous intermediate state of each cell, and assign their ranks accordingly. FTP is easy to implement. However, it is not scalable due to the overhead required for measuring and storing the resolutions.

One scalable approach is to use HP. In this approach, we use both target threshold voltage as well as reference threshold voltage to represent different ranks. The method can be used to program at least 3 ranks in an SLC, and 7 ranks in an MLC. For simplicity, we illustrate HP for SLCs. To program a length-3 permutation, the cell with rank 1 is programmed using the programming command, making the threshold voltage of the cell reach the target threshold voltage of level 0. The cell with rank 2 is programmed using multiple PPs until the cell starts being read 0. The threshold voltage of the cell will be right above the reference threshold voltage between levels 1 and 0. The cell with rank 3 is untouched. To read the permutation stored, we first read each cell with the reference threshold voltage, the cell which reads 1 is given rank 3. For the other two cells which read 0, we issue a programming command on each of them, measuring the time cost. The cell which takes less time to be programmed is given rank 1, and the other is given rank 2.

Note that, both emulation schemes use destructive reading—when permutation is read, the cell states will be changed, and the stored permutation can not be read again. With the help of the advanced testing commands provided by Toshiba, such problem can be resolved by reading using multiple reference thresholds instead of using programming.

10.3 Preliminary Results

We implemented both emulation schemes methods within Micron 34nm SLC flash chips. As a first step, we considered an interference-free case where only the even (or odd) page in a wordline is used to store permutations. Two pages are separated by a wordline. In each page, we use the first 1024 bytes to store 1024 length-3 random permutations, and only the first 3 of the 8 cells belonging to a byte are utilized for each permutation.

For FTP, around 15 permutations on average were misread in each page with the misread probability being 1%. HP performs significantly better. On average, around 5 permutations were misread with misread probability being 0.5%. The major error patterns include both erasures and errors. An erasure happens when two cells have the same rank, e.g. $(1, 2, 3) \to (1, 1, 2)$. An error happens when the output ranks of two cells switch, e.g. $(1, 2, 3) \to (2, 1, 3)$. Table 10.2 lists all the error patterns encountered in each emulation method. HP also yields simpler error patterns.

Table 10.2: Error patterns of the two emulation schemes.

	Erasure	**Error**
FTP	ranks 1 and 2, ranks 2 and 3	ranks 1 and 2, ranks 2 and 3
HP	ranks 1 and 2	ranks 2 and 3

To see how misread probability changes with program/erase cycle, we further cycle the flash chips by repeatively programming pseudo-random length-3 permutations into multiple pages of a block using HP, and then erasing the block. Before an erasure, the permutations are read and compared with the input permutations to compute the average misread probability at each PEC. The cases with and without cell-to-cell interference were examined. In the case without interference, only the even page in each wordline is used, and every two wordlines that store permutations are separated by another unused wordline. The misread probabilities are shown in Figure 10.4. The results suggest that erasure be the dominant error pattern, and the erasure rates increase with PEC. This is due to the program/erase cycling noise which makes the rank 2 cells be overprogrammed more easily during the iterative partial programming process of HP. In the case with interference, each page in the block is used for storing permutations. Therefore, when a page is being programmed, the cells in the adjacent pages will be interfered, and when a page is being read, the adjacent pages will again be interfered due to the programming operation used during the destructive reading process. Our experiments compared the performance of parallel and sequential HP. Parallel HP programs the cells in the same page that are given the same

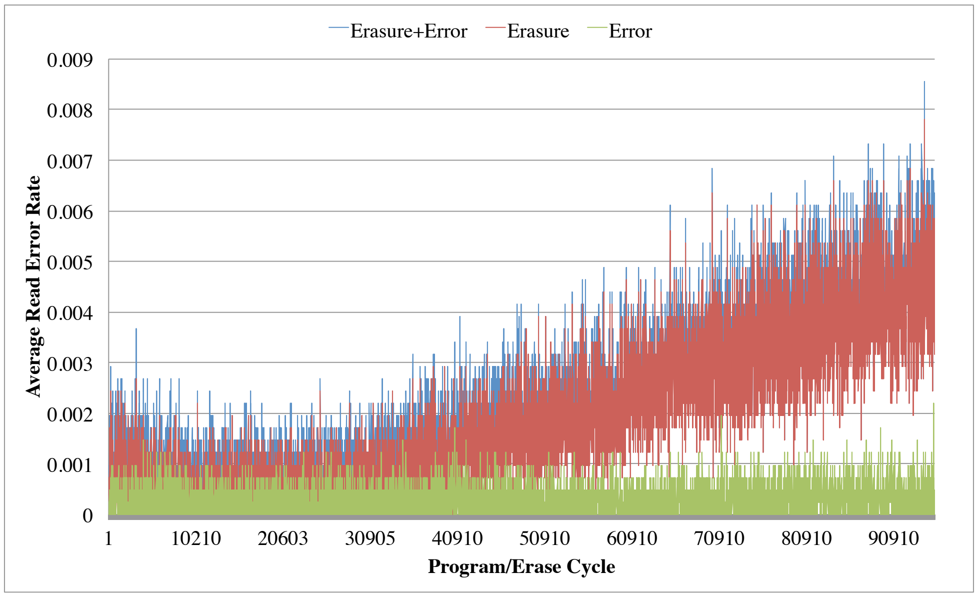


Figure 10.4: The average read error probabilities at different PECs without cell-to-cell interference.

rank in parallel. Therefore, length-3 permutations can be programmed into a page in two steps by first programming the cells of rank 1, and then the cells of rank 2. The sequential HP programs each cell in a page sequentially. The average read error rates in this case are shown in Figure 10.5. The misread probabilities of the case with interference are significantly higher than those of the previous case without interference. By using parallel programming, the two-step HP yields much better performance than the sequential HP does as much less interference is introduced thanks to parallel programming.

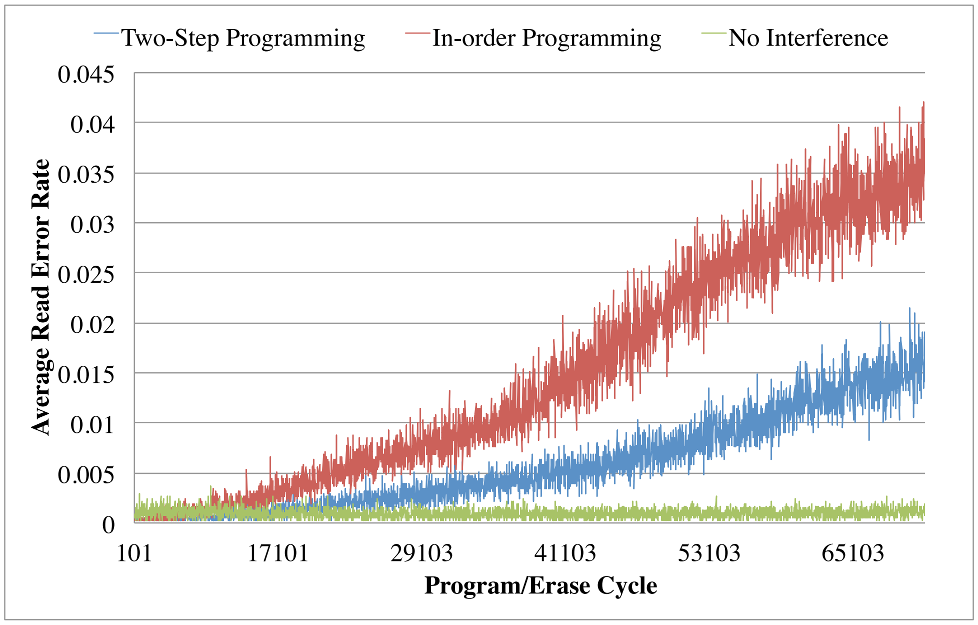


Figure 10.5: The average read error probabilities at different PECs with cell-to-cell interference.

11. SUMMARY AND DIRECTIONS OF FUTURE WORK

There are several research problems that we would like to explore in the future.

We would like to further extend the study of the bit-fixing coding scheme discussed in Chapter 3. Since bit-fixing codes can work with any numeral systems, one direction is to apply bit-fixing codes to the rank modulation scheme [39] whose codewords rely on the factoradic numeral system. Since bit-fixing codes are good at correcting asymmetric errors, another direction would be to see how it performs to correct errors in PCMs against resistance drift. It would also be interesting to compare the performance of bit-fixing coding schemes with a symmetric ECC using dynamic thresholds for quantizing cell levels. This requires us to derive the code rates for each of the scheme given the channel statistics of PCMs, as well as running simulations to see the experimental performance.

One major tasks left for the ECC-WOM codes presented in Chapter 5 and Chapter 7 is to simulate the real error correction performance of the WOM codes. Currently the simulator has been implemented, therefore, the task is relatively easy. After obtaining results, we could further prepare a journal version of the paper. Another main direction is to extend the current code construction to support q-ary codes. To do so, we can start from the noise free q-ary Polar WOM codes studied by Burshtein [10] and try to adapt our construction from binary case to q-ary case. Again, after the code construction is clear, experiments need to be done to test its performance.

In flash memories, various kinds of errors will be introduced during the data retention period. For aged cells, the errors accumulate even more quickly. One simple way for preventing error accumulation is to use memory scrubbing. That is, codewords stored in memory cells are read and decoded periodically. The corrected codewords are written back to the cells. The problem with the basic scrubbing scheme is that frequent cell reprogramming may introduce many block erasures, and degrade writing speed as well as cell quality. With the ECC-WOM codes [38] studied by us, memory scrubbing can be done more efficiently in the sense that the bits to be rewritten to cells are WOM codewords. WOM codes will significantly reduce the number of block erasures, and thus have great potential to increasing the error scrubbing performance. We plan to study different kinds of constructions for memory scrubbing codes, based on which lower bounds of the achievable rates need be derived. The derivation should be based on the previous results on the code rates derived for the ECC-WOM codes. Simulations will need to be carried out to evaluate the experimental performance of the scheme.

One interesting question to answer for content-assisted decoding is that will this approach work for any kind of data? For instance, will this approach work for image files stored in some format such as JPEG? To find the solution, one direction we could possibly study is how to estimate the probability distribution of the output symbols of a source encoder. Once such information is known, we could utilize such information to improve the performance of channel decoding feeding the probability distribution to a soft decoder.

The final objective of emulating and characterizing rank modulation is to show the superiority of rank modulation over the traditional storage scheme.

Currently, we are trying to emulate rank modulation codes in MLC chips. After which, there are many directions that we plan to explore: (1) study methods for further improve the misread probability of the emulation schemes, (2) explore the emulation scheme which combines both FTP and HP to program more than 3 ranks in a SLC and 7 ranks in MLC, (3) characterize the errors on rank modulation under different kinds of noise in flash, which includes program/erase cycling noise, charge leakage (retention), and cell-to-cell interference, (4) compare the capacity and the endurance of rank modulation with traditional SLC and MLC flash memories, (5) propose extensions to multipermutation such that more than one cell can have the same rank, and (6) test on the SLC and MLC flash chips from different major chip manufacturers.

To further make rank modulation codes practical, error correction codes need to be constructed for the scheme. One direction to approach the problem is to adapt the code construction by Barg and Mazumdar [6] studied for rank modulation based on permutations to multi-rank modulation scheme based on multiset permutation. If the adaptation can be done successfully, we could further apply similar asymptotic code rate analysis done in [6] to analyze the performance of the new codes.

The size of data being generated in every day's computation activities is becoming larger and larger. The big data size introduces extraordinary time complexity for retrieving the data that are needed by user. One way towards a better data retrieval methods would be to take full advantage of the most attractive features of NVMs—fast random access capability. One idea is to implement the concept called content-addressable memories. In such scheme, data in memories are organized by the meanings of their contents. When some specific kind of data is needed by user, he/she provides some hint as query, and we will return all the data that are related to the query. To achieve so, We need to design algorithms for comparing the semantics and contents between data, and study the algorithms which search for related data given a set of queries. The complexities of the algorithms can be further studied. Since this application does not require to modify the internal structure of memory chips, it is possible for us to build a real data retrieval application using software based, and its usefulness can be evaluated using real world data sets.

REFERENCES

[1] R. Ahlswede, H. Aydinian, and L. Khachatrian. Unidirectional error control codes and related combinatorial problems. *Proceedings of Eight International Workshop on Algebraic and Combinatorial Coding Theory*, pages 6–9, 2002.

[2] E. Arıkan. Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels. *IEEE Trans. Inf. Theor.*, 55(7):3051–3073, July 2009.

[3] E. Arikan. Source polarization. In *Proceedings of IEEE International Symposium on Information Theory*, pages 899–903, 2010.

[4] E. Arikan. Systematic polar coding. *IEEE Communications Letters*, 15(8):860–862, 2011.

[5] A. Bandyopadhyay, G. J. Serrano, and P. Hasler. Programming analog computational memory elements to 0.2% accuracy over 3.5 decades using a predictive method. In *Proceedings of IEEE International Symposium on Circuits and Systems*, volume 3, pages 2148–2151, May 2005.

[6] A. Barg and A. Mazumdar. Codes in permutations and error correction for rank modulation. *IEEE Transactions on Information Theory*, 56(7):3158–3165, July 2010.

[7] V. Bohossian, A. Jiang, and J. Bruck. Buffer coding for asymmetric multi-level memory. In *Proc. IEEE International Symposium on Information Theory*, pages 1186 –1190, June 2007.

[8] G. W. Burr, M. J. Breitwisch, M. Franceschini, et al. Phase change memory technology. *J. Vac. Sci. Technol. B*, 28(2):223–262, 2010.

[9] D. Burshtein and A. Strugatski. Polar write once memory codes. In *Proc. IEEE International Symposium on Information Theory*, pages 1972–1976, July 2012.

[10] D. Burshtein and A. Strugatski. Polar write once memory codes. *IEEE Transactions on Information Theory*, 59(8):5088–5101, August 2013.

[11] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai. Error patterns in mlc nand flash memory: Measurement, characterization, and analysis. In *Proceedings of Design, Automation Test in Europe Conference Exhibition*, pages 521–526, March 2012.

[12] Y. Cai, E. F. Haratsch, O. Mutulu, and K. Mai. Threshold voltage distribution in mlc nand flash memory: characterization, analysis, and modeling. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1285–1290, 2013.

[13] Y. Cassuto, M. Schwartz, V. Bohossian, and J. Bruck. Codes for asymmetric limited-magnitude errors with application to multilevel flash memories. *IEEE Trans. on Information Theory*, 56(4):1582–1595, April 2007.

[14] G. Cohen, P. Godlewski, and F. Merkx. Linear binary code for write-once memories. *IEEE Trans. Inf. Theor.*, 32(5):697–700, September 1986.

[15] T. M. Cover and J. A. Thomas. *Elements of information theory (2. ed.)*. Wiley, New York, NY, USA, 2006.

[16] N. Elarief and B. Bose. Optimal, systematic, q-ary codes correcting all asymmetric and symmetric errors of limited magnitude. *IEEE Trans. Inf. Theor.*, 56(3):979–983, March 2010.

[17] E. En Gad, A. Jiang, and J. Bruck. Trade-offs between instantaneous and total capacity in multi-cell flash memories. In *Proc. IEEE International Symposium on Information Theory Proceedings*, pages 990–994, July 2012.

[18] E. En Gad, Y. Li, J. Kliewer, M. Langberg, A. Jiang, and J. Bruck. Polar coding for noisy write-once memories. submitted to IEEE International Symposium on Information Theory 2014.

[19] E. En Gad, E. Yaakobi, A. Jiang, and J. Bruck. Rank-modulation rewriting codes for flash memories. In *Proceedings of IEEE International Symposium on Information Theory*, pages 704–708, July 2013.

[20] A. Eslami and H. Pishro-Nik. A practical approach to polar codes. In *Proc. ISIT*, pages 16–20, 2011.

[21] F. Farnoud, V. Skachek, and O. Milenkovic. Error-correction in flash memories via codes in the ulam metric. *IEEE Transactions on Information Theory*, 59(5):3003–3020, May 2013.

[22] M. Franceschini, L.A. Lastras-Montano, A. Jagmohan, M. Sharma, R. Cheek, and Ming-Hsiu Lee. A communication-theoretic approach to phase change storage. In *Proc. IEEE International Conference on Communications (ICC)*, pages 1–6, May 2010.

[23] 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. Inf. Theor.*, 45(1):308–313, September 2006.

[24] L. M. Grupp, J. D. Davis, and S. Swanson. The bleak future of nand flash memory. In *Proceedings of the 10th USENIX conference on File and Storage Technologies*, Berkeley, CA, USA, 2012.

[25] Project Gutenberg. Project Gutenberg, May 2012. http://www.gutenberg.org/.

[26] S. H. Hassani and R. L. Urbanke. Universal polar codes. *CoRR*, abs/1307.7223, 2013.

[27] C. Heegard. On the capacity of permanent memory. *IEEE Trans. Inf. Theor.*, 31(1):34–42, January 1985.

[28] J. Honda and H. Yamamoto. Polar coding without alphabet extension for asymmetric models. *IEEE Transactions on Information Theory*, 59(12):7829–7838, 2013.

[29] N. Hussami, S. B. Korada, and R. Urbanke. Performance of polar codes for channel and source coding. In *Proc. ISIT*, pages 1488–1492, 2009.

[30] D. Ielmini, D. Sharma, S. Lavizzari, and A.L. Lacaita. Physical mechanism and temperature acceleration of relaxation effects in phase-change memory cells. In *Proc. IEEE International Reliability Physics Symposium (IRPS)*, pages 597–603, Phoenix, USA, May 2008.

[31] H. Imai and S. Hirakawa. A new multilevel coding method using error-correcting codes. *IEEE Transactions on Information Theory*, 23(3):371–377, May 1977.

[32] Open NAND Flash Interface. 2012. http://www.onfi.org/.

[33] A.R. Iyengar, P.H. Siegel, and J.K. Wolf. Ldpc codes for the cascaded bsc-bawgn channel. In *47th Annual Allerton Conference on Communication, Control, and Computing*, pages 620–627, 2009.

[34] A. Jiang, V. Bohossian, and J. Bruck. Floating codes for joint information storage in write asymmetric memories. In *Proc. IEEE International Symposium on Information Theory*, pages 1166 –1170, June 2007.

[35] A. Jiang, M. Langberg, M. Schwartz, and J. Bruck. Trajectory codes for flash memory. *IEEE Transactions on Information Theory*, 59(7):4530–4541, July 2013.

[36] A. Jiang, Y. Li, and J. Bruck. Bit-fixing codes for multi-level cells. In *Proc. IEEE Information Theory Workshop (ITW)*, September 2012.

[37] A. Jiang, Y. Li, E. En Gad, M. Langberg, and J. Bruck. Error correction codes for flash memories. In *Proc. Information Theory and Applications Workshop (ITA)*, Feb. 2013.

[38] A. Jiang, Y. Li, E. En Gad, M. Langberg, and J. Bruck. Joint rewriting and error correction in write-once memories. In *Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on*, pages 1067–1071, July 2013.

[39] A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck. Rank modulation for flash memories. *IEEE Transactions on Information Theory*, 55(6):2659–2673, June 2009.

[40] A. Jiang, M. Schwartz, and J. Bruck. Error-correcting codes for rank modulation. In *Proc. IEEE International Symposium on Information Theory*, pages 1736–1740, July 2008.

[41] A. Jiang, M. Schwartz, and J. Bruck. Correcting charge-constrained errors in the rank-modulation scheme. *IEEE Transactions on Information Theory*, 56(5):2112–2120, May 2010.

[42] A. Jiang and Y. Wang. Rank modulation with multiplicity. In *Proc. IEEE GLOBECOM Workshops*, pages 1866–1870, Dec. 2010.

[43] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[44] T. Klove, B. Bose, and N. Elarief. Systematic, single limited magnitude error correcting codes for flash memories. *IEEE Trans. Inf. Theor.*, 57(7):4477–4487, July 2011.

[45] S. B. Korada and R. L. Urbanke. Polar codes are optimal for lossy source coding. *IEEE Trans. Inf. Theor.*, 56(4):1751–1768, April 2010.

[46] C. Leroux, I. Tal, A. Vardy, and W. J. Gross. Hardware architectures for successive cancellation decoding of polar codes. In *IEEE International Conference on Acoustics, Speech and Signal Processing,*, pages 1665–1668, 2011.

[47] Y. Li, A. Hakim, E. F. Haratsch, and A. Jiang. A study of polar codes for flash memories. submitted to IEEE International Symposium on Information Theory 2014.

[48] Y. Li, A. Jiang, and J. Bruck. Error correction and partial information rewriting for flash memories. submitted to IEEE International Symposium on Information Theory 2014.

[49] Y. Li, Y. Wang, A. Jiang, and J. Bruck. Content-assisted file decoding for nonvolatile memories. In *Proc. 46th Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA, 2012.

[50] S. Lin and D. J. Costello. *Error Control Coding, Second Edition*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2004.

[51] D. J. C. MacKay. Good error-correcting codes based on very sparse matrices. *Information Theory, IEEE Transactions on*, 45(2):399 –431, mar 1999.

[52] H. Mahdavifar and A. Vardy. Achieving the secrecy capacity of wiretap channels using polar codes. *IEEE Transactions on Information Theory*, 57(10):6428 –6443, Oct. 2011.

[53] Matt Mahoney. Large Text Compression Benchmark, May 2012. http://mattmahoney.net/dc/text.html.

[54] F. Merkx. Womcodes constructed with projective geometries,. *Traitement du Signal*, 1(2-2):227–231, 1984.

[55] R. Mori and T. Tanaka. Performance of polar codes with the construction using density evolution. *IEEE Communications Letters*, 13(7):519–521, 2009.

[56] A. Pamuk and E. Arikan. A two phase successive cancellation decoder architecture for polar codes. In *Proc. ISIT*, pages 957–961, 2013.

[57] N. Papandreou, H. Pozidis, G. F. Close, et al. Drift-tolerant multilevel phase-change memory. In *Proc. 3rd IEEE International Memory Workshop (IMW)*, pages 1–4, Monterey, USA, may 2011.

[58] W. W. Peterson and E. J. Weldon. *Error-Correcting Codes*. MIT Press, Cambridge, MA, 2nd edition, 1972.

[59] Matt Powell. The Canterbury Corpus, May 2012. http://corpus.canterbury.ac.nz/.

[60] T. Richardson and R. Urbanke. *Modern Coding Theory*. Cambridge University Press, New York, NY, USA, 2008.

[61] R. L. Rivest and A. Shamir. How to reuse a write-once memory. *Information and Control*, 55(1-3):1–19, 1982.

[62] W. Ryan and S. Lin. *Channel Codes, Classical and Modern*. Cambridge University Press, Cambridge, UK, 2009.

[63] F. Sala, R. Gabrys, and L. Dolecek. Dynamic threshold schemes for multi-level non-volatile memories. *IEEE Transactions on Communications*, 61(7):2624–2634, July 2013.

[64] S. Schechter, G. H. Loh, K. Straus, and D. Burger. Use ecp, not ecc, for hard failures in resistive memories. In *Proceedings of the 37th annual international symposium on Computer architecture*, ISCA '10, pages 141–152, New York, NY, USA, 2010. ACM.

[65] D. Shin, S. Lim, and K. Yang. Design of length-compatible polar codes based on the reduction of polarizing matrices. *IEEE Transactions on Communications*, 61(7):2593–2599, 2013.

[66] A. Shpilka. Capacity achieving multiwrite wom codes. *CoRR*, abs/1209.1128, 2012.

[67] A. Shpilka. Capacity achieving two-write wom codes. In *LATIN 2012: Theoretical Informatics*, volume 7256 of *Lecture Notes in Computer Science*, pages 631–642. Springer Berlin Heidelberg, 2012.

[68] I. Tal and A. Vardy. List decoding of polar codes. In *Proc. ISIT*, pages 1–5, 2011.

[69] I. Tal and A. Vardy. How to construct polar codes. *IEEE Transactions on Information Theory*, 59(10):6562–6582, Oct 2013.

[70] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, april 1967.

[71] U. Wachsmann, R. F H Fischer, and J.B. Huber. Multilevel codes: theoretical concepts and practical design rules. *IEEE Transactions on Information Theory*, 45(5):1361–1391, Jul 1999.

[72] J. Wang, T. Courtade, H. Shankar, and R. D. Wesel. Soft information for ldpc decoding in flash: Mutual-information optimized quantization. In *Proc. IEEE Global Telecommunications Conference (GLOBECOM)*, pages 1–6, Houston, TX, 2011.

[73] J. Wang, G. Dong, T. A. Courtade, H. Shankar, T. Zhang, and R. D. Wesel. Ldpc decoding with limited-precision soft information in flash memories. *CoRR*, abs/1210.0149, 2012.

[74] Y. Wang, W. Yu, S. Wu, G. Malysa, G. E. Suh, and E. Kan. Flash memory for ubiquitous hardware security functions: True random number generation and device fingerprints. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 33–47, Washington, DC, USA, 2012. IEEE Computer Society.

[75] Y. Wang, W. Yu, S. Q. Xu, E. Kan, and G. E. Suh. Hiding information in flash memory. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 271–285, Washington, DC, USA, 2013. IEEE Computer Society.

[76] Y. Wu. Low complexity codes for writing a write-once memory twice. In *Proc. IEEE International Symposium on Information Theory*, pages 1928–1932, June 2010.

[77] Y. Wu and A. Jiang. Position modulation code for rewriting write-once memories. *IEEE Trans. Inf. Theor.*, 57(6):3692–3697, June 2011.

[78] W. Xu and T. Zhang. Using time-aware memory sensing to address resistance drift issue in multi-level phase change memory. In *Proc. 11th International Symposium on Quality Electronic Design (ISQED)*, pages 356 –361, San Jose, CA, march 2010.

[79] E. Yaakobi, A. Jiang, and J. Bruck. In-memory computing of akers logic array. In *Proc. IEEE International Symposium on Information Theory*, pages 2369–2373, July 2013.

[80] E. Yaakobi, S. Kayser, P. H. Siegel, A. Vardy, and J. K. Wolf. Efficient two-write wom-codes. In *Proc. IEEE Information Theory Workshop*, pages 1–5, September 2010.

[81] E. Yaakobi, S. Kayser, P. H. Siegel, A. Vardy, and J. K. Wolf. Codes for write-once memories. *IEEE Trans. Inf. Theor.*, 58(9):5985–5999, September 2012.

[82] E. Yaakobi, J. Ma, L. Grupp, P. H. Siegel, S. Swanson, and J. K. Wolf. Error characterization and coding schemes for flash memories. In *Proc. IEEE GLOBECOM Workshops*, pages 1856–1860, December 2010.

[83] E. Yaakobi and A. Shpilka. High sum-rate three-write and non-binary wom codes. In *Proc. IEEE International Symposium on Information Theory*, pages 1386–1390, July 2012.

[84] E. Yaakobi, P. H. Siegel, A. Vardy, and J. K. Wolf. On codes that correct asymmetric errors with graded magnitude distribution. In *Proc. IEEE International Symposium on Information Theory Proceedings*, pages 1056–1060, August 2011.

[85] E. Yaakobi, P.H. Siegel, A. Vardy, and J.K. Wolf. Multiple error-correcting wom-codes. *IEEE Trans. Inf. Theor.*, 58(4):2220–2230, April 2012.

[86] G. Zemor and G. D. Cohen. Error-correcting wom-codes. *IEEE Trans. Inf. Theor.*, 37(3):730–734, May 1991.

[87] F. Zhang, H.D. Pfister, and A. Jiang. Ldpc codes for rank modulation in flash memories. In *Proceedings of IEEE International Symposium on Information Theory*, pages 859–863, June 2010.

[88] H. Zhou, A. Jiang, and J. Bruck. Error-correcting schemes with dynamic thresholds in nonvolatile memories. In *Proc. IEEE International Symposium on Information Theory*, pages 2143 –2147, Saint Petersburg, Russia, 31 2011-aug. 5 2011.

[89] H. Zhou, A. Jiang, and J. Bruck. Systematic error-correcting codes for rank modulation. In *IEEE International Symposium on Information Theory*, pages 2978–2982, July 2012.